Metode de planificare timp real pentru sisteme distribuite cu niveluri mixte de criticalitate

Teză destinată obținerii
titlului științific de doctor inginer
la
Universitatea Politehnica Timișoara
în domeniul Calculatoare și Tehnologia Informației
de către

Ing. Eugenia Ana CAPOTA

Președintele comisiei:	prof.univ.dr.ing. Vladimir I. CREȚU
Conducător științific:	prof.univ.dr.habil.ing. Mihai V. MICEA
Referenți științifici:	prof.univ.dr.ing. Vasile I. MANTA
	prof.univ.dr. Dana PETCU
	prof.univ.dr.ing. Daniel I. CURIAC

Ziua susținerii tezei:

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Inginerie Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică și Telecomunicații
8. Inginerie Industrială
9. Inginerie mecanică
10. Știința Calculatoarelor
11. Știința și Ingineria Materialelor
12. Ingineria Sistemelor
13. Inginerie Energetică
14. Calculatoare și Tehnologia Informației
15. Ingineria Materialelor
16. Inginerie și Management
17. Arhitectură
18. Inginerie Civilă și Instalații
19. Inginerie Electronică, Telecomunicații și Tehnologii Informaționale

Universitatea Politehnica Timișoara a inițiat seriile de mai sus în scopul diseminării expertizei, cunoștințelor și rezultatelor cercetărilor întreprinse în cadrul Școlii doctorale a universității. Seriile conțin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susținute în universitate începând cu 1 octombrie 2006.

România, 300223 Timișoara, Bd. Vasile Pârvan 2B
Tel./fax 0256 404677
e-mail: editura@upt.ro

Cuvânt înainte

Teza de doctorat a fost elaborată pe parcursul activității mele în cadrul Departamentului de Calculatoare și Tehnologia Informației al Universității Politehnica Timișoara.

Mulțumiri deosebite se cuvin conducătorului de doctorat prof.univ.dr.habil.ing. Mihai V. Micea pentru sprijinul acordat și îndrumarea continuă și plină de răbdare, atât în procesul de cercetare, cât și în cadrul carierei mele profesionale, în calitate de asistent de cercetare.

În egală măsură, le aduc mulțumirile mele și membrilor comisiei de îndrumare, prof.univ.dr.ing. Mircea Stratulat, șl.dr.ing. Cristina S. Stângaciu și șl.dr.ing. Valentin Stângaciu pentru sprijinul de specialitate acordat și observațiile riguroase cu privire la documentele elaborate pe parcursul programului de doctorat. De asemenea, un loc aparte în perioada de cercetare l-au avut prof.univ.dr.ing. Daniel I. Curiac și șl.dr.ing. Cristina S. Stângaciu care m-au direcționat și ajutat în procesul de redactare și publicare a lucrărilor științifice. Le mulțumesc tuturor pentru colaborare și suport.

Nu în ultimul rând, aș vrea să mulțumesc în special familiei mele pentru încurajările, încrederea și sprijinul acordat pe parcursul acestui proces de cercetare.

Timișoara, septembrie 2021 Eugenia Ana Capota

Destinatarii dedicației.

Familiei mele.

CAPOTA, Eugenia Ana

Metode de planificare timp real pentru sisteme distribuite cu niveluri mixte de criticalitate

Teze de doctorat ale UPT, Seria X, Nr. YY, Editura Politehnica, 200Z, 114 pagini, 30 figuri, 23 tabele.

ISSN:
ISBN:

Cuvinte cheie
sisteme timp real, sisteme distribuite, sisteme cu niveluri mixte de criticalitate, algoritm de planificare

Rezumat
Sistemele critice sunt prezente în viața de zi cu zi, fiind utilizate în numeroase domenii, de la echipamente medicale și vehicule autonome, până la aplicații militare. Acestea implică, pe de o parte, un răspuns în timp real datorită interacțiunii directe cu mediul înconjurător și, pe de altă parte, includerea mai multor funcționalități critice. Teza de față prezintă evoluția modelelor de task-uri și a algoritmilor de planificare pentru sisteme cu niveluri mixte de criticalitate și posibilitatea integrării acestui concept în sistemele cyber physical. De asemenea, un nou model de task-uri este propus pentru platforme distribuite, precum și un algoritm de planificare perfect periodic pentru sisteme cu mai multe unități de procesare și niveluri mixte de criticalitate.

CUPRINS

NOTAȚII, ABREVIERI, ACRONIME

LISTA DE TABELE

LISTA DE FIGURI

1 INTRODUCERE

1.1 Domeniul abordat și motivația cercetării

Sistemele de timp real (RT: Real-Time) sunt prezente în viața de zi cu zi, fiind utilizate în numeroase domenii, de la echipamente medicale și vehicule autonome, până la aplicații militare. Aceste tipuri de sisteme implică, pe de o parte, un răspuns în timp real datorită interacțiunii directe cu mediul înconjurător și, pe de altă parte, o administrare eficientă a resurselor.

Un concept care prezintă interes ridicat în sistemele de timp real clasice este includerea mai multor funcționalități critice în cadrul aceleași platforme. Astfel, s-a introdus o nouă clasă de sisteme, și anume sistemele cu niveluri mixte de criticalitate (MCSs: Mixed Criticality Systems). Acest concept se referă la *„o platformă computațională incorporată, în care aplicații de criticalitate diferită împart resurse computaționale și/sau de comunicare”* [1].

Deși sistemele cu niveluri mixte de criticalitate au apărut inițial în automotive și avionică, aplicabilitatea lor se extinde și la alte domenii, precum: IoT (Internet of Things) [2, 3], dispozitive medicale [3], sisteme industriale [4] și sisteme cyber physical în general.

Multe aplicații de timp real critice au fost deja implementate folosind arhitecturi distribuite eterogene [5]. Aceste arhitecturi conțin o colecție de componente independente, împărțite pe mai multe unități de procesare și care comunică între ele prin intermediul unei rețele [6].

Un domeniu vast care utilizează arhitecturi distribuite eterogene este reprezentat de sistemele cyber physical (CPSs: Cyber Physical Systems), din care fac parte atât platformele robotice, cât și sistemele inteligente din automotive și avionică. Aceste sisteme includ în modelul lor *„o funcție logică, astfel încât orice modificare a stării mediului exterior va altera starea funcției logice și/sau orice modificare a stării funcției logice va conduce la schimbarea stării mediului exterior”* [7].

Integrarea conceptului de MCSs în CPSs necesită o abordare specifică datorită cerințelor care pot să apară ca fiind contradictorii uneori. Restricțiile existente asupra resurselor, precum și constrângerile ce țin de putere și cost trebuie luate în considerare pentru îndeplinirea cerințelor de siguranță și de timp real, ceea ce introduce un grad ridicat de complexitate în cazul CPSs.

Planificarea în CPSs pe baza conceptului de MCSs s-a realizat, în mare parte, la nivel de aplicație. Acest lucru a dus la dezvoltarea unor modele de task-uri particulare variate, cu o multitudine de constrângeri impuse de comportamentul sistemului [8]. Deși relevante din punct de vedere practic, aceste metode prezintă neajunsuri când vine vorba de scalabilitatea platformei și interacțiunea cu un mediu imprevizibil.

Pentru facilitarea și integrare eficientă a MCSs in CPSs trebuie identificate particularitățile celor din urmă, precum și influența lor asupra diferiților algoritmi de planificare, pornind de la un model de task-uri standardizat.

Un alt domeniu care extinde aplicabilitatea sistemelor distribuite eterogene este IoT. Într-o era a telecomunicațiilor și a interconectivității, IoT este o tehnologie nouă, ce oferă posibilitatea de a conecta dispozitivele inteligente care ne înconjoară, formând o rețea distribuită peste regiuni geografice întinse. Acest lucru oferă posibilitatea de a dezvolta servicii și aplicații inovatoare [9].

Arhitecturile IoT integrează diferite tipuri de componente care folosesc diverse concepte, metodologii și tehnologii, unele relativ noi, dar cu un rol important în cadrul acestor platforme. Sistemele de timp real, precum și cele care nu sunt de timp real (Non-RT: Non-Real-Time), dar și sistemele critice și cele care nu sunt critice, coexistă, având cereri din ce în ce mai complexe (uneori chiar și contradictorii) de la rețea. Astfel, este necesară dezvoltarea unor concepte noi pentru aprovizionarea, administrarea și monitorizarea acestor sisteme și a componentelor lor. Dezvoltarea conceptelor respective a dus la apariția unui nou domeniu, cel al sistemelor IoT de timp real (RT-IoT: Real-Time Internet of Things) [10]. RT-IoT promite o experiență mai bună pentru utilizatori datorită conexiunii în timp real, ceea ce duce la utilizarea eficientă a dispozitivelor integrate din generațiile viitoare care prezintă cerințe de timp tot mai stricte.

Conceptul de sisteme cu niveluri mixte de criticalitate poate fi aplicat și în RT-IoT, ceea ce duce la apariția unei noi paradigme de planificare, și anume sisteme IoT cu niveluri mixte de criticalitate (MC-IoT: Mixed Criticality Internet of Things).

Nevoia de soluții care să integreze sisteme cu restricții sau cerințe stricte de timp în arhitecturile IoT a fost explicată pe larg în [11]. Această nevoie se reflectă și în încercările de a dezvolta și implementa protocoale de comunicație de timp real, „time-triggered" care oferă suport pentru platforme critice [5]. Cu toate acestea, puține sisteme de timp real clasice sau sisteme cu niveluri mixte de criticalitate, au fost incorporate în arhitecturi IoT. Câteva exemple de aplicații și dispozitive IoT autonome sunt prezentate în [3] și [12], iar în [13] este descris un exemplu de sistem IoT colaborativ omogen. O platformă care vizează reducerea complexității când vine vorba de dezvoltarea aplicațiilor critice compatibile cu IoT a fost introdusă în [2]. Datorită capacității de a rula o varietate mare de aplicații care prezintă diferite niveluri de criticalitate și cerințe de timp variate (de exemplu rețele de senzori pentru aplicații de conducere autonome [14], clădiri inteligente [15], sau aplicații medicale critice [16]), sistemele MC-IoT pot să abordeze eficient problema administrării resurselor în platformele IoT. Problema planificării în sistemele de timp real distribuite eterogene nu a fost intensiv cercetată, în ciuda potențialului pe care îl oferă [17]. Câteva exemple de lucrări din literatura de specialitate sunt [18-20]. În [18] este realizat un studiu cu privire la sistemele de operare dezvoltate pentru dispozitive IoT, care scoate în evidență caracteristicile necesare ale unui astfel de sistem de operare. Apoi, este propus un nou model de planificare [19] în cadrul căruia aplicațiile RT-IoT au nevoie de mai multe tipuri de date pentru a lua decizii, maximizând astfel calitatea informațiilor colectate. Întrucât rețelele fără fir de timp real sunt esențiale pentru aplicațiile IoT, [20] introduce un nou cadru de planificare dinamică distribuită a pachetelor, care să reducă numărul de pachete pierdute și să asigure gestionarea eficientă a evenimentelor critice din rețea.

Caracterul eterogen al arhitecturii IoT influențează în mod direct aplicațiile care rulează pe diferitele componente hardware:

- În primul rând, timpul de execuție al codului aplicației este influențat puternic de sistemul pe care rulează aplicația.
- În mod asemănător, consumul de energie electrică cauzat de execuția aceluiași cod, poate să difere pe sistemul k, față de sistemul j.
- Uneori este posibil ca unele aplicații să nu poată să ruleze deloc pe o anumită platformă.

Din această cauză, este nevoie de modele noi pentru aplicațiile MC-IoT care să ia în considerare particularitățile hardware, pe lângă aspectele temporale și de criticalitate.

În ultima secțiune a tezei, accentul se pune pe o categorie specială de aplicații critice care rulează într-un mediu controlat de timp, și necesită planificarea perfect periodică (fără jitter) a anumitor task-uri critice. Această nevoie poate să apară în urma unor probleme de sincronizare a mesajelor [21], datorită aplicațiilor de procesare a semnalelor [22-25], sau din cauza unor condiții impuse de diferite tipuri de certificări [26]. În plus, pentru orice sistem incorporat, este de preferat ca unele task-uri să fie executate fără jitter, întrucât prezența acestuia introduce dificultăți în buclele de control [26].

Execuția task-urilor într-un mediu controlat de timp pentru sistemele clasice de timp real se realizează utilizând tehnici de planificare time-triggered, dintre care se remarcă abordarea statică table-driven.

Abordarea table-driven e bazată pe analiza statică a planificării, prin generarea unui tabel (tabelul planificării) care va fi folosit la rulare, pentru a decide momentul de timp la care instanța unui task (denumită și job) trebuie să își înceapă execuția [27].

Deși sistemele clasice de timp real implică construirea unui singur tabel al planificării, în MCSs lucrurile devin mai complexe datorită prezenței mai multor moduri de rulare. Trecerea de la un mod de criticalitate la altul corespunde cu tranziția de la un tabel al planificării la altul. Deci, în MCSs fiecărui nivel de criticalitate îi corespunde un tabel al planificării [28].

MCSs pot fi folosite pentru a oferi un răspuns în timp real, izolând în același timp funcționalitățile critice ale platformei. Dacă analizăm sistemele critice, în cazul MCSs există unele avantaje ale abordărilor table-driven, față de abordările event-driven (bazate pe evenimente): certificare mai ușoară, deoarece abordările table-driven sunt complet deterministe [28]; posibilă sincronizare între task-uri [21]; gestionare eficientă a energiei electrice, deoarece fiecare mod de rulare corespunde unui nivel de criticalitate, și fiecare nivel folosește propriul tabel al planificării; dar și adaptarea mai ușoară a aplicațiilor de timp real din diferite domenii precum automotive, avionică, etc., care folosesc deja metode table-driven [29].

1.2 Obiectivele tezei de doctorat

Scopul acestei teze de doctorat este de a dezvolta un model de task-uri standardizat pentru sisteme de timp real distribuite cu niveluri mixte de criticalitate care să ia în considerare particularitățile hardware ale platformei pe care rulează, dar și de a introduce un algoritm de planificare perfect periodică pentru sisteme cu mai multe unități de procesare și niveluri mixte de criticalitate.

Astfel, primul obiectiv al cercetării de doctorat [O1] a presupus:

[O1]. Studiul și analiza stadiului actual al domeniului sistemelor cu niveluri mixte de criticalitate, cu accent pe sistemele distribuite.

Obiectivul a fost concretizat prin parcurgerea următoarelor etape:

[T1.1]. Compararea diferitelor modele de task-uri existente.

[T1.2]. Clasificarea algoritmilor de planificare principali în funcție de arhitectura platformei pe care rulează.

Activitatea aferentă primului obiectiv [O1] a fost parțial publicată și în [A2].

Al doilea obiectiv al programului doctoral [O2] a fost sintetizat printr-un model de task-uri pentru sisteme de timp real distribuite cu niveluri mixte de criticalitate:

[O2]. Dezvoltarea unui model de task-uri.

Pentru atingerea acestui obiectiv au fost îndeplinite următoarele sarcini:

[T2.1]. Definirea unui model de task-uri pentru sisteme de timp real distribuite cu niveluri mixte de criticalitate.

[T2.2]. Implementarea modelului de task-uri pe un simulator.

[T2.3]. Testarea modelului de task-uri propus anterior, pe simulator, cu ajutorul algoritmilor de planificare.

Obiectivul [O2] este inclus într-un articol în curs de publicare.

Al treilea obiectiv al cercetării de doctorat [O3] a fost realizat prin introducerea unui algoritm de planificare pentru sisteme de timp real cu niveluri mixte de criticalitate:

[O3]. Dezvoltarea unui mecanism de planificare.

Pentru a îndeplini acest obiectiv au fost parcurse două etape:

[T3.1]. Realizarea unui algoritm de planificare pentru sisteme de timp real cu niveluri mixte de criticalitate.

[T3.2]. Conceperea unui test de planificare pentru algoritm.

Această activitate [O3] a fost publicată și în [A3].

De asemenea, ca ultim obiectiv al programului doctoral s-a urmărit [O4]:

[O4]. Testarea și validarea algoritmului.

Obiectivul a cuprins următoarea sarcină de lucru:

[T4.1]. Testarea, validarea și compararea diferitelor versiuni pentru algoritmul definit la pasul anterior cu ajutorul unui simulator.

1.3 Structura tezei de doctorat

Capitolul 1 cuprinde o introducere în domeniul sistemelor de timp real cu niveluri mixte de criticalitate, obiectivele vizate ale cercetării de doctorat și structura curentă a tezei.

Următorul capitol (Capitolul 2) prezintă evoluția modelelor de task-uri, dar și o clasificare a algoritmilor de planificare pentru sisteme cu niveluri mixte de criticalitate din literatura de specialitate.

Capitolul 3 este împărțit în două secțiuni: prima tratează domeniul sistemelor cyber physical, iar a doua prezintă platformele IoT. Algoritmii de planificare clasificați anterior sunt comparați pe baza unor caracteristici comune, indiferent de domeniu, ale sistemelor cyber physical (Secțiunea 3.1.2). Provocările și avantajele integrării conceptului de niveluri mixte de criticalitate în sistemele cyber physical sunt evidențiate în Secțiunea 3.1.3. Următoarea secțiune (Secțiunea 3.2.1) analizează aspectele teoretice din lucrările de specialitate ce abordează domeniul IoT. Pe baza acestora, se poate realiza o formalizare matematică a problemei planificării în sistemele de timp real distribuite. În Secțiunea 3.2.2 este propusă o arhitectură MC-IoT pentru sisteme de timp real distribuite cu niveluri mixte de criticalitate.

În cadrul acestei arhitecturi este definit un nou model de task-uri pentru sisteme distribuite cu niveluri mixte de criticalitate (Capitolul 4). Acesta extinde modelul clasic pentru sisteme cu niveluri mixte de criticalitate, astfel încât resursele platformei să fie administrate cât mai eficient.

În continuare, este introdusă o metodologie de mapare a task-urilor pe diferite platforme care să ia în considerare atât particularitățile temporale, cât și hardware. Metodologia cuprinde diferite tehnici de stabilire a gradului de afinitate a fiecărui task pentru un anumit element de procesare, și de definire a unei funcții de mapare adecvată, astfel încât să fie respectate cerințele aplicației și constrângerile ce țin de resurse.

Prin implementarea și testarea acestora în cadrul unui mediu de simulare (Capitolul 5), tehnicile de stabilire a afinității, precum și metodologia de mapare a task-urilor sunt evaluate în Capitolul 6.

Următoarea secțiune (Capitolul 7) descrie algoritmul propus pentru sisteme cu o singură unitate de procesare, cu niveluri mixte de criticalitate.

Metoda de planificare introdusă poartă denumirea de FENP_MC (Fixed Execution Non-Preemptive Mixed Criticality), fiind un algoritm de timp real, table-driven, non-preemptiv, adaptat pentru sisteme cu niveluri mixte de criticalitate, conform tehnicii FENP (Fixed Execution Non-Preemptive) pentru sisteme clasice de timp real, care garantează o execuție perfect periodică (fără jitter), într-un mediu controlat de timp.

Apoi, în Capitolul 8, pe baza acestui algoritm, este propusă o metodă de planificare pentru sisteme omogene cu mai multe unități de procesare, denumită P_FENP_MC (Partitioned Fixed Execution Non-Preemptive Mixed Criticality), folosind o euristică de partiționare. De asemenea, sunt introduse teste de fezabilitate, atât pentru sisteme cu o singură unitate de procesare, cât și pentru sisteme cu mai multe unități de procesare.

În ultima parte a tezei (Capitolul 9) este realizată o analiză a performanței algoritmului dezvoltat, prin compararea cu alte metode de planificare, într-un context non-preemptiv. Rezultatele acestor comparații evidențiază rata de succes și jitter-ului de planificare, prin implementarea și testarea acestor algoritmi în cadrul unui mediu de simulare.

Capitolul 10 cuprinde concluziile, contribuțiile personale și perspectivele viitoare de dezvoltare.

1.4 Principalele contribuții

Principalele contribuții aduse în această teză sunt:

- identificarea unor caracteristici comune ale sistemelor cyber physical și evaluarea algoritmilor de planificare pentru sisteme cu niveluri mixte de criticalitate din literatura de specialitate, pe baza acestor atribute, atât la nivel de unitate de procesare, cât și pentru sisteme cu mai multe unități de procesare.
- definirea unui model de task-uri pentru sisteme distribuite eterogene cu niveluri mixte de criticalitate și introducerea unei metodologii de mapare a task-urilor în cadrul acestor platforme.
- implementarea unui algoritm de planificare perfect periodică, table-driven, non-preemptiv pentru sisteme cu mai multe unități de procesare, cu niveluri mixte de criticalitate.

2 STADIUL ACTUAL ÎN DOMENIUL SISTEMELOR CU NIVELURI MIXTE DE CRITICALITATE

2.1 Definirea sistemelor cu niveluri mixte de criticalitate și clasificarea acestora

Conceptul de sistem cu niveluri mixte de criticalitate a apărut pentru prima dată în lucrarea lui Vestal din 2007 [30]. Acesta consideră un sistem alcătuit din componente cu două sau mai multe niveluri de criticalitate diferite (de exemplu safety-critical – scenarii ce țin de siguranța personală și a mediului înconjurător, mission-critical – se referă la capacitatea de a finaliza obiectivele pentru care sistemul a fost dezvoltat și non-critical). Criticalitatea unei componente determină nivelul de rigurozitate aplicat în proiectarea și analiza funcționalității, dar și în determinarea cantității de resurse utilizate [31]. Într-un astfel de sistem, timpul de execuție pentru cazul cel mai defavorabil devine dependent de nivelul de criticalitate al componentei de care aparține.

Pentru a simplifica lucrurile, în literatura de specialitate se consideră sisteme cu două niveluri de criticalitate [1]. Cu toate acestea, de cele mai multe ori, modelul poate fi extins la mai multe moduri de rulare. Câteva lucrări care tratează platforme cu cel puțin trei niveluri de criticalitate sunt [32-34].

Într-un sistem cu niveluri mixte de criticalitate aplicațiile pot fi văzute ca un ansamblu de procese (task-uri). O trecere de la un mod de rulare la altul duce la abandonarea task-urilor de criticalitate mai scăzută, în timp de task-urile de criticalitate ridicată sunt executate utilizând parametrii corespunzători noului mod de rulare al sistemului [35-38].

Abandonarea task-urilor de criticalitate scăzută duce la degradarea serviciilor oferite de aceste platforme și la irosirea capabilităților de procesare. Pentru a evita acest lucru au fost introduse diverse soluții care permit continuarea execuției task-urilor de criticalitate scăzută, precum:

- Rularea tuturor task-urilor, cu extinderea perioadelor și/sau a deadline-urilor, metodă denumită „elastic task model” [39-41].
- Rularea tuturor task-urilor, cu reducerea timpilor de execuție pentru task-urile de criticalitate scăzută [42].
- Abandonarea instanțelor care aparțin de un anumit subset de task-uri [43, 44].
- Finalizarea execuției instanțelor de criticalitate scăzută numai pentru cazul în care rulează în momentul trecerii de la un nivel de criticalitate la altul (instanțele de criticalitate scăzută care nu sunt în execuție vor fi abandonate) [45].
- Aplicarea conceptului de „online adaptive task dropping”, în care task-urile de criticalitate scăzută sunt abandonate selectiv, în timpul rulării sistemului [46].
- Utilizarea așa numitului „slack time”, definit ca fiind diferența temporală între deadline-ul absolut al unei instanțe, timpul de activare și timpul de execuție alocat de către sistem. Acesta poate fi utilizat pentru a executa task-uri de criticalitate scăzută după schimbarea modului de rulare al sistemului [32, 47, 48].

- În cazul sistemelor cu mai multe unități de procesare, schimbarea modului de rulare pentru o unitate de procesare poate rezulta în migrarea task-urilor de criticalitate scăzută pe altă unitate de procesare pentru care modul de rulare nu a fost modificat [49].

În următoarea secțiune este prezentată o evoluție a principalelor modele de task-uri pentru sisteme cu niveluri mixte de criticalitate din literatura de specialitate.

2.2 Modele de task-uri pentru sisteme cu niveluri mixte de criticalitate

Modul în care componentele software sau funcționalitățile sistemului sunt modelate constituie principala diferență între domeniile sistemelor de timp real. Pentru unele zone de aplicabilitate există procese care rulează în permanență, în timp ce în cadrul altor domenii, funcționalități multiple ale sistemului rulează ca entități software diferite pe același suport hardware, sau rulează ca servicii software [50].

Task-urile sunt unitățile de execuție de bază ale unei aplicații. Acestea pot să fie periodice sau sporadice, în funcție de modul de activare. Task-urile periodice sunt activate la o rată constantă, în timp ce task-urile sporadice au un interval (minim) de timp între două instanțe consecutive ale aceluiași task [51, 52]. Vestal a propus un model pentru MCSs [30], având la bază modelul de task-uri sporadice introdus de Mok, în 1983. Astfel, în sistemele cu niveluri mixte de criticalitate fiecărui task îi este atribuit un nivel de criticalitate și un set de proprietăți [30, 53]:

$$\tau_i = \left\{T_i, D_i, L_i, \left\{C_{i,L_j} \mid j \in 1 \dots l\right\}\right\} \tag{2-1}$$

unde:

- l este numărul de niveluri de criticalitate;
- T_i reprezintă perioada pentru task-urile perioadice, sau intervalul (minim) de timp între două instanțe consecutive ale aceluiași task i pentru task-urile sporadice;
- D_i indică momentul până la care trebuie încetată execuția unei instanțe, relativ la timpul de activare;
- L_i este nivelul de criticalitate (1 fiind nivelul cel mai scăzut);
- C_{i,L_j} reprezintă timpul de execuție (vector de valori – câte o valoare pentru fiecare nivel de criticalitate mai mic sau egal cu L_i, exprimând timpul de execuție pentru cazul cel mai defavorabil pentru fiecare nivel de criticalitate).

Un task constă dintr-o serie de instanțe (job-uri), fiecare job moștenind setul de parametrii al task-ului (T_i, D_i, L_i), la care adaugă proprii parametri [54]. Astfel, job-ul k al task-ului τ_i este caracterizat de:

$$J_{i,k} = \left\{a_{i,k}, d_{i,k}, c_{i,k}, T_i, D_i, L_i\right\} \tag{2-2}$$

unde:

- $a_{i,k}$ este timpul de activare ($a_{i,k+1} - a_{i,k} \geq T_i$);
- $d_{i,k}$ semnifică deadline-ul absolut ($d_{i,k+1} = a_{i,k} + D_i$);
- $c_{i,k}$ reprezintă timpul de execuție alocat de către sistem, care este dependent de modul de rulare al sistemului (pentru L_j, $c_{i,k} = C_{i,L_j}$);

- T_i, D_i, L_i au aceeași semnificație ca în (2-1).

Pe baza acestui model de task-uri și a altor extensii au fost dezvoltați numeroși algoritmi de planificare, în cele mai multe cazuri particulari, orientați spre anumite aplicații.

O variantă generică a modelului lui Vestal, utilizată în numeroase cercetări, acceptă vectori de valori pentru următorii parametri: perioadă, deadline și timp de execuție. În acest caz, atât T_i, cât și D_i sunt dependente de nivelul de criticalitate, nu doar C_i.

Acest model a stat la baza altor versiuni mai simplificate, precum cel introdus de Burns în [55]. Diferența principală este dată de timpul de execuție, care în acest caz conține două valori $C_i(SELF)$ și $C_i(NORMAL)$. $C_i(SELF)$ sau $C_i(SF)$ se referă la nivelul de criticalitate al task-ului i, iar $C_i(NORMAL)$ sau $C_i(NL)$ reprezintă timpul de execuție pentru nivelul de criticalitate cel mai scăzut din întregul sistem. Astfel, fiecare instanță este caracterizată de un vector cu două valori pentru timpul de execuție, indiferent de numărul de niveluri de criticalitate distincte ale sistemului.

O altă simplificare clasică a modelului consideră doar două niveluri de criticalitate (Lo și Hi) [42].

Accesul la memorie pentru cazul cel mai defavorabil este adăugat la modelul general în [56].

Un alt model este descris în [57], denumit E-MC (Elastic Mixed-Criticality), prin care se introduce ideea de perioadă variabilă pentru task-uri de criticalitate scăzută. Task-urile de criticalitate ridicată sunt modelate la fel ca în exemplele precedente, principala diferență fiind în reprezentarea task-urilor de criticalitate scăzută, care au intervalul corespunzător perioadei exprimat cu ajutorul a doi parametri (perioada maximă și perioada dorită, care este aceeași cu perioada din modelul clasic).

Pe lângă modelele clasice folosite în MCSs, [50] propune un model pentru sisteme cyber physical cu niveluri mixte de criticalitate. Autorii evidențiază importanța task-urilor soft de timp real, cu un DM (Deadline Miss) tolerabil al instanțelor în sistemele cyber physical cu niveluri mixte de criticalitate și prezintă un model pentru aceste tipuri de task-uri. Modelul poate fi văzut ca o transformare a celui descris în [58] pentru task-uri de timp real, într-un model pentru MCSs, prin exprimarea numărului de DMs tolerabile ca funcție a nivelului de criticalitate, în locul unei valori constante, asemenea modelului clasic de timp real.

Pentru seturile de task-uri care prezintă constrângeri de prioritate există modele derivate din modelele de task-uri de timp real bazate pe grafuri, precum cel propus în [59]. Acestea sunt mai complexe, deoarece descriu, pe lângă comportamentul temporal, și dependențele de funcționalitate dintre task-uri. Modelul de task-uri MS-DRT (Mode-Switching Diagraph Real-Time), introdus de Ekberg și Yi [60], este un asemenea exemplu.

Un alt model bazat pe grafuri, care specifică nivelul de interferență permis între task-uri, a fost propus în [44], și anume ICG (Interference Constraint Graph). Parametrii task-urilor sunt aceeași cu cei introduși de Vestal, principala diferență fiind dată de faptul că graful modelează interferența dintre task-uri.

Avantajul modelelor bazate pe grafuri este că permit utilizarea unor seturi de task-uri complexe, care nu sunt neapărat independente, ceea ce le face o soluție viabilă pentru modelarea unei execuții dependente a instanțelor. Un alt avantaj reprezintă modelarea unitară, care conține atât comportamentul sistemului din punctul de vedere al planificării, cât și comportamentul temporal al task-urilor.

Dezavantajul acestor metode este complexitatea, fiind mai dificil de utilizat la analiza diferiților algoritmi de planificare.

Figura 1 prezintă principalele modele de task-uri pentru MCSs, diferențele dintre acestea, precum și evoluția lor din RTSs (Real-Time Systems). Cel mai cunoscut este modelul lui Vestal (2007), care are la bază lucrarea lui Mok din 1983 pentru task-uri sporadice. Pornind de la acesta, au fost dezvoltate o serie de modele simplificate, precum cel introdus de Burns și Baruah (2013) sau modelul lui Burns (2015). Pe lângă modelele de task-uri generale, există și câteva specifice pentru task-uri soft de timp real cu niveluri mixte de criticalitate, ca de exemplu modelul din lucrarea lui Lee și Shin (2017). Adițional, sunt evidențiate și modelele bazate pe grafuri, care de asemenea prezintă variațiuni: modelul lui Ekberg și Yi (2016) și modelul lui Huang et al. (2013). Săgețile din figură semnifică generalizări sau simplificări ale unor modele.

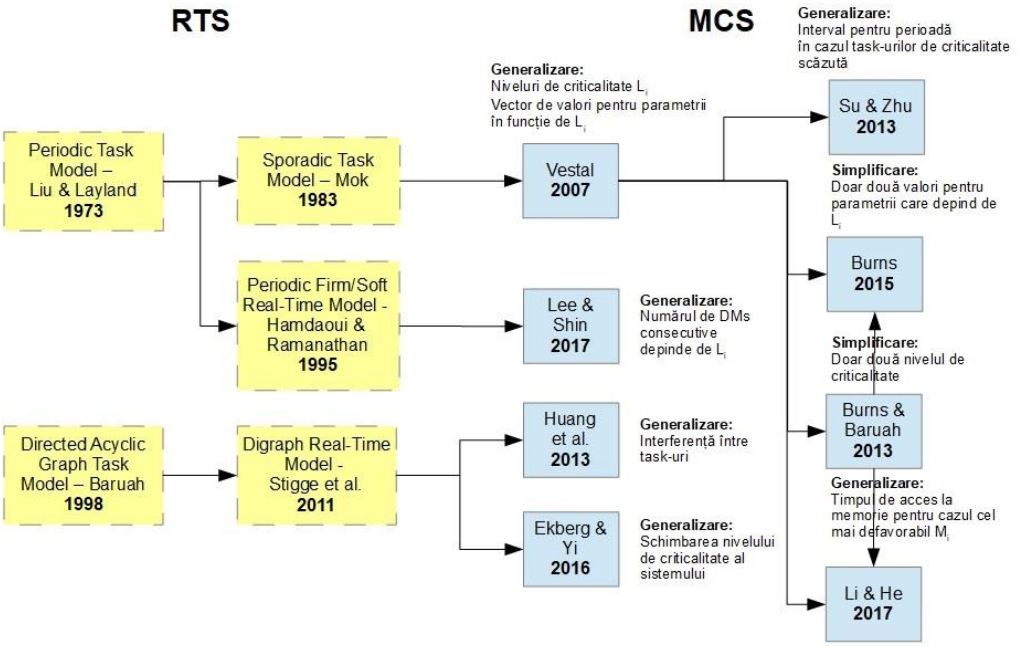


Figura 1. Evoluția modelelor de task-uri.

Chiar dacă există un număr mare de modele de task-uri, majoritatea sunt doar variațiuni provenite din câteva modele de bază. Acestea, deși compatibile între ele, se concentrează pe diferite aspecte, generalizând sau simplificând anumite comportamente sau parametri.

2.3 Planificarea în sisteme cu niveluri mixte de criticalitate

În sistemele de timp real, atingerea performanței impuse sistemului din punctul de vedere al comportamentului temporal se realizează prin planificarea task-urilor. Pentru aplicațiile cu niveluri mixte de criticalitate, lucrurile devin mai complicate datorită creșterii numărului de variabile ce trebuie luate în considerare [61].

Asemenea sistemelor de timp real, algoritmii de planificare pentru MCSs se împart în algoritmi pentru sisteme cu o singură unitate de procesare și algoritmi

pentru sisteme cu mai multe unități de procesare, în funcție de platforma vizată. Pe de altă parte, există algoritmi statici, dacă deciziile ce țin de planificare se fac offline (înainte de începerea rulării sistemului), algoritmi dinamici, dacă deciziile ce țin de planificare sunt luate în timpul rulării sistemului, la diferite momente de timp și algoritmi hibrizi, în care procesul decizional este atât online, cât și offline.

În continuare vor fi prezentați principalii algoritmi de planificare atât pentru sisteme cu o singură unitate de procesare, cât și pentru sisteme cu mai multe unități de procesare în MCSs.

2.3.1 Planificarea la nivel de unitate de procesare

A. Clasificare

Tehnicile de planificare în sistemele cu o singură unitate de procesare pot fi clasificate în funcție de modul în care prioritatea este atribuită instanțelor unui task [62]:

- Clasa Fixed Task-Priority (FTP) – toate instanțele generate de un task au aceeași prioritate.
- Clasa Fixed Job-Priority (FJP) – instanțe diferite ale aceluiași task pot avea priorități diferite, dar nu este permisă schimbarea priorității unei instanțe între momentul de start și momentul de stop al acesteia.
- Clasa Dynamic Priority (DP) – prioritățile instanțelor se pot modifica între momentul de start și momentul de stop.
- Clasa Hybrid Priority (HP) – politicile de planificare prezintă caracteristici specifice mai multor clase de algoritmi.

Planificarea FJP este o generalizare a planificării FTP, iar DP este o generalizare a planificării FJP.

Acești algoritmi au abordări diferite când vine vorba de atribuirea priorității instanțelor și dezvoltarea unor teste de planificare eficiente.

B. Atribuirea priorității task-urilor/instanțelor

Atribuirea priorității diferitelor instanțe ale unui task trebuie să ia în considerare, pe lângă parametrii temporali (de exemplu perioadă sau deadline), și nivelul de criticalitate a task-ului respectiv. În acest sens au fost propuse câteva soluții [38]:

- Folosirea nivelurilor de criticalitate pentru atribuirea priorității: criticality monotonic priority assignment sau Own Criticality Based Priority (OCBP) [38, 62]. În cadrul acestei metode, toate instanțele de criticalitate ridicată au o prioritate mai mare decât orice instanță de criticalitate scăzută. Pentru instanțele care au același nivel de criticalitate, prioritățile sunt atribuite conform unei scheme optime standard, precum deadline rate monotonic priority assignment.
- Utilizarea unei tehnici, a cărui concept provine din sistemele clasice de timp real, denumită Period Transformation (PT). Această metodă transformă perioada inițială a task-ului, astfel încât să reflecte atât comportamentul temporal, cât și nivelul de criticalitate al task-ului respectiv. În acest mod prioritățile unor instanțe de criticalitate diferită sunt intercalate, de exemplu EDF-VD [63] și EDF-DB [64].
- Aplicarea algoritmilor Static Mixed Criticality (SMC) și Adaptive Mixed Criticality (AMC), pe baza metodei Audsley de atribuire a priorităților [62].
- Utilizarea unor euristici pentru atribuirea priorităților [61].

C. Teste de planificare

Majoritatea algoritmilor, atât pentru MCSs statice, cât și dinamice, au fost testați aplicând Response Time Analysis (RTA) [8], mai ales în cazul atribuirii statice a priorităților, dar și Demand Bound Function (DBF) [65], în cazul atribuirii dinamice a priorităților. Adițional, pentru algoritmii de planificare preemptivi cu prioritate fixă, care rulează pe platforme cu o singură unitate de procesare, un test de planificare exact este prezentat în [61].

Aceste teste analizează planificarea în diferite moduri de criticalitate, dar nu și în timpul schimbării modului de criticalitate al sistemului.

Următorul tabel este preluat din [17], în cadrul căruia am făcut o clasificare a principalilor algoritmi pentru sisteme cu o singură unitate de procesare existenți în literatura de specialitate, scoțând în evidență asemănările și deosebirile dintre aceștia.

Tabel 1. Algoritmi de planificare pentru sisteme cu o singură unitate de procesare.

Algoritmul de planificare	**Clasa**	**Tipul procesului de atribuire a priorității**	**Metoda de atribuire a priorității**	**Implementarea în literatura de specialitate**	**Testul de planificare**	**Tipul testului de planifi-care**	**Ref.**
Fixed Task-Priority Static Mixed Criticality (FTP-SMC)	FTP	static	Audsley	Simulator/Linux	Bazat pe RTA: SMC-NO	suficient	[8], [38], [66]
Fixed Task-Priority Adaptive Mixed Criticality (FTP-AMC)	FTP	static	Audsley	Simulator/Linux	Bazat pe RTA: AMC-RT AMC-MAX	suficient	[8], [38], [66]
Fixed Task-Priority preemptive (FTP-preemptive)	FTP	static	euristică	Simulator	SB-RTA	exact	[61]
Zero-Slack Scheduling (ZSS)	FTP	static	RM	Simulator/Linux	RTA/Bazat pe Slack	suficient	[66], [67]
Fixed Job-Priority Own Criticality Based Priority (FJP-OCBP)	FJP	static	Audsley	Teoretic	Bazat pe DBF	suficient	[66], [68], [69], [70]
Earliest Deadline First with Virtual Deadlines (EDF-VD)	FJP	dinamic	EDF	Simulator/Linux	Bazat pe DBF	suficient	[8], [55], [63], [66], [71], [72]
Earliest Deadline First with Demand Bound (EDF-DB)	FJP	dinamic	EDF	Simulator/Linux	Bazat pe DBF	suficient	[64], [66]

Earliest Deadline First with Adaptive Task Dropping (EDF-AD)	FJP	dinamic	EDF	Simulator	Bazat pe DBF	suficient	[73]
Earliest Deadline First with Adaptive Task Dropping-Enhanced (EDF-AD-E)	FJP	dinamic	EDF	Simulator	Bazat pe DBF	suficient	[73]
Criticality Based Earliest Deadline First (CBEDF)	DP	dinamic	OCPB	Simulator	Bazat pe DBF	suficient	[74]
Priority List Reuse Scheduling (PLRS)	FJP	hibrid	OCBP	Simulator	Bazat pe DBF	suficient	[66], [75]

Figura de mai jos este o reprezentare grafică a algoritmilor clasificați în Tabel 1, având la bază modelul ilustrat în [75] pentru sisteme de timp real clasice. Săgețile sunt folosite pentru a simboliza faptul că unii algoritmi au la bază o metodă de planificare deja existentă. Chenarele colorate cu galben evidențiază tehnicile de planificare clasice, iar cele colorate cu albastru ilustrează algoritmii dezvoltați pentru sisteme cu niveluri mixte de criticalitate.

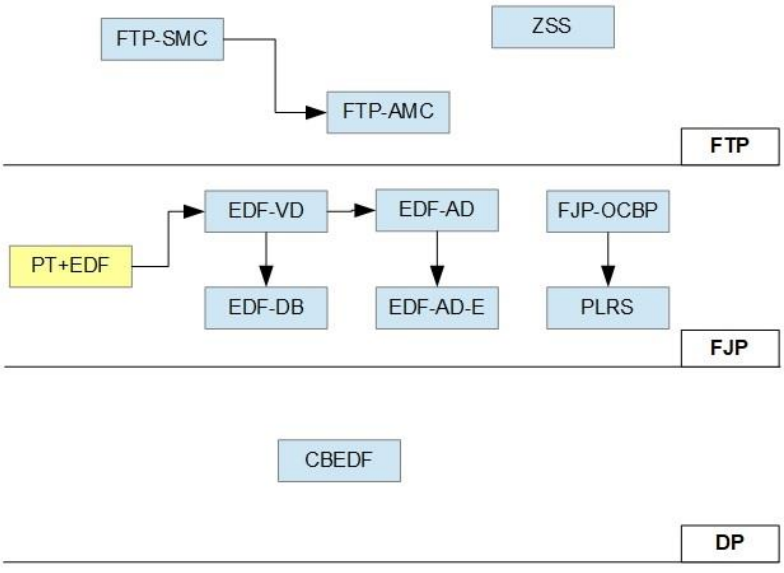


Figura 2. Clasificarea algoritmilor de planificare pentru sisteme cu o singură unitate de procesare.

2.3.2 Planificarea în sisteme cu mai multe unități de procesare

A. Clasa P: Algoritmi de planificare partiționați

În cazul algoritmilor de planificare partiționați, fiecare task este alocat unei singure unități de procesare. Câteva avantaje ale acestei metode de planificare sunt [62]:

- Depășirea deadline-ului unui task poate să afecteze doar task-urile aflate pe aceeași unitate de procesare.
- Nu există penalizare în cazul migrării.
- Fiecare unitate de procesare folosește o coadă de rulare separată care ajută la reducerea overhead-ului.

Pe de altă parte există o serie de dezavantaje ale planificării partiționate [62]:

- Unele seturi de task-uri sunt planificabile doar dacă este permisă migrarea între unitățile de procesare.
- Fiecare set de task-uri are nevoie de o nouă alocare optimă la unitatea de procesare corespunzătoare.
- Unitățile de procesare sunt de multe ori inactive. Acestea nu pot fi folosite de către task-urile deja alocate la o unitate de procesare diferită.

Numeroase studii abordează cercetarea algoritmilor de planificare partiționați atât în sistemele de timp real clasice [76], cât și în MCSs. Baruah et al. [77] a descris și evaluat un set de algoritmi pentru planificarea partiționată pe mai multe unități de procesare, și anume MC-PARTITION. De asemenea, au fost dezvoltate câteva îmbunătățiri empirice: MC-PARTITION-UT-0.75, MC-PARTITION-UT-1 și MC-PARTITION-UT-INC. Ultima versiune domină primele două. Acest algoritm, precum și îmbunătățirile sale, utilizează EDF-VD ca tehnică de planificare la nivel de unitate de procesare.

Alți doi algoritmi de partiționare pentru sisteme de timp real sunt prezentați în [78]: Time Triggered Scheduler with Mode Change (TTS-MC) și Event Scheduler in Multi-Core. Pentru TTS-MC fiecare unitate de procesare are propriul Time Triggered Scheduler, care se ocupă doar de planificarea task-urilor alocate acesteia. În mod asemănător, pentru Event Scheduler-MC, fiecare unitate de procesare are propriul Event Scheduler.

O metodă de planificare adițională pentru platforme partiționate este descrisă în [79-81]. Tehnica MC-IS-Server extinde strategia IS-Server, prezentată în cadrul acelorași studii, la MCSs. Algoritmul poate fi aplicat sistemelor cu două sau mai multe niveluri de criticalitate. O euristică de partiționare a task-urilor și de scurtare a deadline-urilor este folosită, și anume MC-EY-WF (Mixed-Critical EY Worst Fit). Conform tehnicii, sunt utilizate două servere globale: unul pentru task-urile de criticalitate ridicată „Hi” și unul pentru task-urile de criticalitate scăzută „Lo”. Când sistemul este în modul Lo, task-urile sunt planificate în cadrul fiecărui server conform tehnicii partiționate EDF, folosind deadline-uri scurtate pentru task-urile de criticalitate Hi. În momentul în care sistemul intră în modul Hi, serverul aferent task-urilor de criticalitate Lo este dezactivat, iar task-urile rămase de criticalitate Hi sunt planificate aplicând metoda partiționată EDF. În acest caz se folosesc deadline-urile originale ale task-urilor.

B. Clasa G: Algoritmi de planificare globali

Algoritmii de planificare ce aparțin acestei clase permit migrarea task-urilor între unitățile de procesare. Principalele avantaje ale planificării globale sunt [62]:

- Mai puține comutări de context, deoarece o preempțiune poate să apară doar atunci când nu există nici o unitate de procesare inactivă.
- Nu este nevoie de algoritmi de alocare atunci când setul de task-uri se modifică.
- Unitățile de procesare sunt utilizate mai eficient datorită faptului că task-urile pot să migreze între acestea.

Cele mai semnificative probleme sunt [62]:

- Migrarea task-urilor între unitățile de procesare este permisă, ceea ce poate să ducă la un overhead ridicat în sistem.
- Folosește o singură coadă de așteptare pentru toate unitățile de procesare.
- Poate necesita utilizarea unei memorii comune și a unor canale de comunicare deoarece planificarea globală crește fluxul de comunicare între unitățile de procesare.
- Predictibilitate scăzută.
- Performanță scăzută în cazul unor seturi de task-uri particulare.

Studiul publicat de Baruah et al. [77] introduce un algoritm care extinde abordarea pentru sisteme cu o singură unitate de procesare prezentată în [69] la platforme cu mai multe unități de procesare, prin aplicarea algoritmului de planificare global fpEDF [82] (dezvoltat pentru sisteme de timp real clasice) în contextul MCSs. De asemenea, o versiune îmbunătățită a algoritmului a fost descrisă în cadrul aceluiași studiu, și anume GLOBAL-PRAGMATIC. În ambele cazuri, task-urile alocate la fiecare unitate de procesare sunt planificate local folosind tehnica EDF-VD.

Lee et al. [36] propune o tehnică de planificare pentru platforme cu mai multe unități de procesare în sisteme cu două niveluri de criticalitate, denumită MC-Fluid. Asemenea metodei EDF-VD, MC-Fluid ia în considerare doar două niveluri de criticalitate (Hi și Lo), unde deadline-urile task-urilor de criticalitate Hi sunt reduse în modul Lo. Atât în modul Hi, cât și în modul Lo, task-urile sunt planificate folosind o altă tehnică, DP-Fair [83]. MC-IS-Fluid [80, 81] extinde algoritmul MC-Fluid pentru a include izolare între diferitele niveluri de criticalitate. În plus oferă suport pentru mai mult de două niveluri de criticalitate.

C. Clasa C: Algoritmi de planificare cluster/semi-partiționați

Un algoritm de planificare cluster/semi-partiționat este o abordare hibridă între algoritmii partiționați și cei globali în care unitățile de procesare sunt grupate în clustere și sub-clustere. Această metodă are următoarele beneficii [84, 85]:

- Task-urile sunt grupate în subseturi care sunt alocate unităților de procesare și executate secvențial, rezultând în zero overhead în interiorul unui cluster.
- Reduce timpul de execuție paralel.
- Task-urile din fiecare cluster pot fi planificate folosind algoritmi de planificare globali diferiți.
- Reduce costurile de migrare deoarece majoritatea task-urilor sunt partiționate folosind planificarea semi-partiționată, iar restul pot să migreze într-o manieră bine administrată.
- Îmbunătățește utilizarea procesorului, spre deosebire de abordarea partiționată.
- Favorizează platforme de dimensiuni mari, cu multiple unități de procesare.

Pe de altă parte, prezintă unele dificultăți [84, 85]:

- Complexitate computațională relativ ridicată.

- Clusterele de dimensiuni mici pot suferi limitări din punctul de vedere al încărcării în modurile de criticalitate ridicată.

Ali și Kim [85] au fost primii care să propună o metodă de planificare bazată pe clustere pentru MCSs cu mai multe unități de procesare. Tehnica folosește clustere de dimensiuni mici (sub-clustere) în modul de criticalitate scăzută pentru că utilizarea fiecărui task este mai mică, și clustere de dimensiuni mari în modul de criticalitate ridicată, datorită creșterii utilizării fiecărui task. Task-urile de criticalitate scăzută se opresc din execuție la schimbarea modului de rulare al sistemului din criticalitate scăzută, în criticalitate ridicată. Toate task-urile din sub-clustere sunt planificate inițial în modul de criticalitate scăzută folosind un algoritm global cu prioritate fixă. Adițional, setul de task-uri este ordonat în mod descrescător cu metoda decreasing criticality – decreasing utilization (DCDU), iar alocarea task-urilor la clustere se realizează folosind euristica Worst Fit [86].

D. Clasa D: Algoritmi de planificare distribuiți

Scalabilitatea este o caracteristică importantă care trebuie considerată în sistemele de timp real cu mai multe unități de procesare, ceea ce duce către o abordare distribuită, deoarece planificarea centralizată nu este fezabilă pentru coordonarea unor componente multiple într-un mediu dinamic.

Algoritmii de planificare distribuiți încapsulează tehnici de planificare atât pentru sisteme cu o singură unitate de procesare, cât și pentru sisteme cu mai multe unități de procesare, deci aceștia pot fi văzuți ca fiind complecși dar practici pentru integrarea capabilităților fizice și computaționale pe aceeași platformă.

Algoritmii de planificare distribuiți [6] sunt dezvoltați pentru sisteme în care există componente împărțite pe mai multe unități de procesare. Principala caracteristică a unui algoritm de planificare distribuit este că folosește middleware distribuit pentru a interconecta partiții. O partiție poate să conțină una sau mai multe unități de procesare.

Principalele dezavantaje ale clasei distribuite sunt:

- Complexitate crescută.
- Probleme la alocarea resurselor adiționale.
- Necesită partiționare spațială și temporală.

În schimb există și o serie de avantaje importante:

- Performanță ridicată.
- Flexibilitate.
- Adaptabilitate.
- Eficiență energetică.

Sistemele de timp real cu mai multe unități de procesare încapsulează componente multiple, cu diferite specificații, cerințe de sistem și configurații hardware, astfel că algoritmii de planificare distribuiți pentru sisteme eterogene aduc o îmbunătățire substanțială în comparație cu abordările centralizate. Acest tip de planificare este optimă, chiar dacă prezintă limitări când vine vorba de partajarea resurselor. Există mai multe forme de eterogenitate: configurațională, care implică cerințe diferite ce țin de aplicație si consum de putere; arhitecturală, cu privire la capabilitățile sistemului și nu în ultimul rând, eterogenitatea sistemului de operare, deoarece este posibil ca unități de procesare diferite să necesite configurații ale sistemului de operare diferite [87].

Un exemplu notabil de cercetare pentru sisteme de timp real cu privire la acest subiect a fost evidențiat de Perez et al. [88]. Lucrarea descrie utilizarea unei platforme de timp real partiționată distribuită care incorporează tehnici hypervisor și middleware distribuit standard. Arhitectura distribuită propusă pentru MCSs cu mai

multe unități de procesare folosește XtratuM [89] ca hypervisor, precum și standardul DDS (Data Distribution Service) [90]. Standardul DSS este un middleware care se bazează pe un model de comunicare de tip publisher-subscriber, în care datele, definite de topic-uri, pot fi transmise între entități publisher sau subscriber în cadrul unui spațiu de date global. Subscriberii trebuie să specifice topic-urile de care sunt interesați pentru a primi datele respective, iar comunicarea este permisă doar între entitățile publisher și subscriber care provin din același domeniu. Metoda hypervisor XtratuM folosește tehnica de para-virtualizare pentru a furniza CPU-uri virtuale partițiilor și poate atribui aceeași politică de planificare la una sau mai multe unități de procesare.

O serie de modele pentru MCSs sunt, de asemenea, dezvoltate în cadrul proiectului european DREAMS [91]. Arhitectura are la bază noduri alcătuite din una sau mai multe partiții cu niveluri mixte de criticalitate. Modelele discutate descriu soluții generice reutilizabile pentru hypervisori, dispozitive multi-core COTS (Commercial-Off-The-Shelf) și rețele cu niveluri mixte de criticalitate.

Un algoritm de tip bin-packing pentru MCSs, denumit COP (Compress-on-Overload Packing) a fost introdus în [92], ca extensie la metoda de planificare ZSRM (Zero-Slack Rate-Monotonic) [93]. Acest algoritm are ca principal scop maximizarea unei metrici denumită ductilitate în platforme de timp real distribuite cu niveluri mixte de criticalitate. Ductilitatea caracterizează comportamentul sistemului în condiții de overload din perspectiva alocării resurselor. Metrica asigură procesarea cu succes a task-urilor de criticalitate ridicată chiar și în condiții de overload.

Asigurarea izolării temporale și spațiale este foarte importantă într-un sistem care conține atât aplicații critice, cât și aplicații care nu sunt critice. Astfel, o tehnică pentru optimizarea sloturilor de timp în sistemele incorporate eterogene cu niveluri mixte de criticalitate a fost propusă în [93]. Abordarea utilizează un model partiționat, fiecare aplicație rulând pe o partiție diferită, în cadrul căreia este alocat un anumit număr de sloturi de timp.

Xie et al. [94] a propus un alt algoritm pentru sisteme incorporate eterogene cu niveluri mixte de criticalitate, și anume D_MHEFT. Scopul, în acest caz, este ca task-urile de criticalitate ridicată să fie executate înainte de deadline, rezultând într-un DMR (Deadline Miss Ratio) scăzut. Modul de criticalitate în care se află sistemul este modificat pentru a planifica eficient și corect task-uri de criticalitate mai mare sau egală cu cea a sistemului.

Două tehnici de planificare eficiente pentru sistemele automotive au fost introduse de Xie et al. [95], și anume algoritmii dinamici de planificare FDS_MIMF și ADS_MIMF. FDS_MIMF a fost dezvoltat cu scopul de a minimiza makespan-urile individuale ale funcțiilor din perspectiva unei performanțe ridicate, pentru a satisface cerințele de eterogenitate, dinamism și paralelism din automotive. ADS_MIMF aduce câteva îmbunătățiri, prin tratarea unor provocări adiționale, precum siguranță și criticalitate. Această metodă permite atingerea unor valori scăzute ale DMR în cazul task-urilor de criticalitate ridicată, menținând simultan o performanță acceptabilă pentru aceste sisteme.

Algoritmii de planificare distribuiți pot fi bazați pe un model partiționat sau global (D_MHEFT, F_MHEFT). Adițional, există două tipuri de planificare: statică (D_MHEFT, F_MHEFT) și dinamică (FDS_MIMF, ADS_MIMF). În cadrul planificării statice, task-urile sunt lansate simultan, iar în cazul planificării dinamice, acestea sunt lansate la momente de timp diferite.

Asemenea clasificării algoritmilor de planificare pentru platforme cu o singură unitate de procesare, următorul tabel, publicat în [17], tratează principalii algoritmi

pentru sisteme cu mai multe unități de procesare existenți în literatura de specialitate, scoțând în evidență asemănările și deosebirile dintre aceștia.

Tabel 2. Algoritmi de planificare pentru sisteme cu mai multe unități de procesare.

Algoritmul de planificare	**Clasa**	**Implementarea în literatura de specialitate**	**Permite migrarea task-urilor**	**Ref.**
Mixed-Criticality-PARTITION (MC-PARTITION)	P	Simulator	Nu	[77]
Mixed-Criticality PARTITION-Utilization-0.75 (MC-PARTITION-UT-0.75)	P	Simulator	Nu	[77]
Mixed-Criticality PARTITION-Utilization-1 (MC-PARTITION-UT-1)	P	Simulator	Nu	[77]
Mixed-Criticality PARTITION-INCREMENT (MC-PARTITION-UT-INC)	P	Simulator	Nu	[77]
Time-Triggered Scheduler with Mode Change (TTS-MC)	P	Simulator/Linux	Nu	[78]
Event Scheduler in Multi-Core	P	Simulator/Linux	Nu	[78]
Mixed-Criticality Isolation Server (MC-IS-Server)	P	Simulator	Nu	[81]
Task Grouping-Partitioned Earliest Deadline First (TG-PEDF)	P	Simulator	Nu	[96]
Notional Processor Scheduling-Fractional Capacity-Integrated Modular Avionics (NPS-F-IMA)	P	Simulator	Nu	[84]
Partitioned-Eckberg (P-EKB)	P	Simulator	Nu	[84]
Partitioned Earliest Deadline First with Virtual Deadlines (P-EDF-VD)	P	Simulator/Linux	Nu	[97]
Flexible Time-Triggered Scheduling (FTTS)	P	Simulator/Platfomă Kalray MPPA-256 Andey many-core	Nu	[96]
Energy Minimized Mixed-Criticality (EM3)	P	Simulator	Nu	[98]
Isolated Mixed-Criticality (IM3)	P	Simulator	Nu	[98]
Abordarea lui Ali și Kim	C	Simulator	Da	[85]
Notional Processor Scheduling-Fractional Capacity-Mixed-Criticality (NPS-F-MC)	C	Simulator	Da	[84]
Semi-Partitioned-Eckberg (SP-EKB)	C	Simulator	Da	[84]
Mixed-Criticality-Reduction to Uniprocessor (MxC-RUN)	G	Simulator	Da	[99]
GLOBAL	G	Simulator	Da	[77]
GLOBAL-PRAGMATIC	G	Simulator	Da	[77]
Mixed-Criticality-Fluid (MC-Fluid)	G	Simulator	Da	[36]
Mixed-Criticality Isolation-Fluid (MC-IS-Fluid)	G	Simulator	Da	[81]
Linear Programming Dynamic Power Management-Mixed-Criticality (LPDPM-MC)	G	Simulator	Da	[100]
Fairness on Multiple Heterogeneous Earliest Finish Time (F_MHEFT)	D	Simulator	Da	[94]
Deadline-span of Multiple Heterogeneous Earliest Finish Time (D_MHEFT)	D	Simulator	Da	[94]

Fairness-based Dynamic Scheduling-Minimize Individual Makespans of Functions (FDS_MIMF)	D	Simulator	Da	[95]
Adaptive Dynamic Scheduling-Minimize Individual Makespans of Functions (ADS_MIMF)	D	Simulator	Da	[95]

Figura 3 este o reprezentare grafică a algoritmilor încadrați în Tabel 2, având la bază modelul ilustrat în [75] pentru sisteme de timp real clasice. Săgețile sunt folosite pentru a sublinia algoritmii care au la bază o metodă de planificare deja existentă. Chenarele colorate cu galben evidențiază metodele de planificare clasice, iar cele colorate cu albastru, algoritmii dezvoltați pentru sisteme cu niveluri mixte de criticalitate.

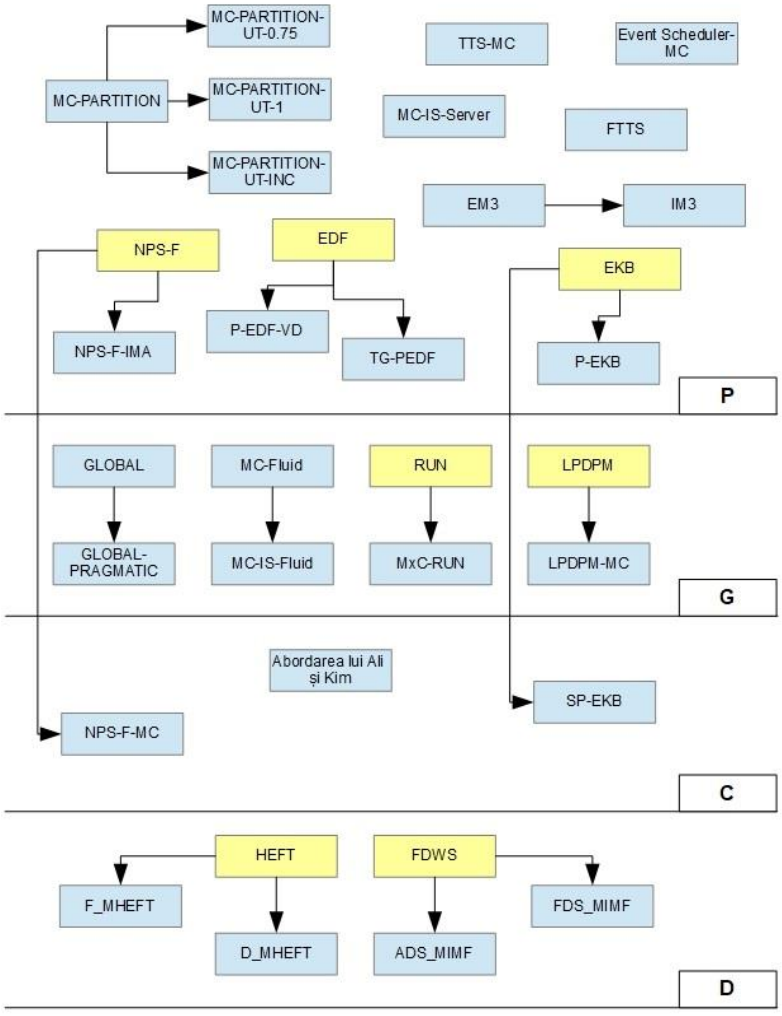


Figura 3. Clasificarea algoritmilor de planificare pentru sisteme cu mai multe unități de procesare.

2.4 Clasificarea algoritmilor în funcție de punctele de planificare

De la formalizarea primului model pentru sisteme cu niveluri mixte de criticalitate – modelul lui Vestal [30], s-a acordat o atenție deosebită acestui domeniu, concretizată printr-o serie de algoritmi de planificare. Acești algoritmi pot fi clasificați în funcție de momentele de timp la care se iau deciziile ce țin de planificare (puncte de planificare), în trei categorii: event-driven, time-triggered și ierarhici.

Pentru algoritmii event-driven, punctele de planificare sunt definite prin evenimente de terminare sau activare a execuției unui task. În cazul algoritmilor time-triggered, deciziile de planificare sunt luate la momente de timp predefinite. Abordările ierarhice combină atât tabele ale planificării, cât și metode de planificare event-driven.

Un studiu amplu privind planificarea în MCSs [31] a arătat că, până recent, problema planificării s-a concentrat mai mult pe abordări event-driven, în ciuda faptului că există progres important cu privire la algoritmii time-triggered și hibrizi. Astfel, câteva exemple de metode event-driven au fost introduse în [46, 74, 75, 101, 102]. Un algoritm event-driven utilizat frecvent în literatura de specialitate este EDF-VD (Earliest Deadline First with Virtual Deadlines) pe două niveluri de criticalitate (Hi – criticalitate ridicată și Lo – criticalitate scăzută) [63]. Această tehnică calculează un deadline virtual pentru fiecare task de criticalitate Hi, dacă sistemul rulează în modul de criticalitate Lo. În modul de criticalitate Hi, task-urile de criticalitate Hi sunt planificate pe baza deadline-ului real. Această modificare se aplică pentru a echilibra planificarea task-urilor pe diferite niveluri de criticalitate, furnizând o mai bună planificare și performanță în timpul rulării sistemului.

Datorită predictibilității, abordările time-triggered au fost utilizate tot mai des în ultimii ani, dar lucrările relevante în domeniu sunt limitate, existând posibilitatea extinderii pe viitor. În ultimul deceniu au apărut câteva lucrări de specialitate care abordează planificarea într-un mediu time-triggered pentru sisteme cu niveluri mixte de criticalitate. Astfel, în [103] a fost prezentată o euristică pentru construirea tabelelor planificării într-un mediu time-triggered. Algoritmul se bazează pe backtracking pentru a ghida căutarea într-o structură de tip arbore, și este format din două euristici: una pentru construirea tabelelor planificării, iar cealaltă pentru realizarea operației de backtracking.

O altă metodă pentru construirea tabelelor planificării, bazată pe prioritate, a fost introdusă în [28]. Tehnica incorporează trecerea de la un mod de rulare la altul (mode-change), ceea ce mărește flexibilitatea și performanța sistemului. În [104], este propus un algoritm de planificare time-triggered, dezvoltat pentru MCSs cu mai multe unități de procesare identice, atât în contextul unor job-uri independente, cât și pentru job-uri dependente. În acest scop sunt construite două tabele ale planificării separate pentru fiecare unitate de procesare: un tabel pentru modul de rulare Lo și un tabel pentru modul de rulare Hi. De asemenea, planificarea este globală, ceea ce înseamnă că job-urile pot migra de pe o unitate de procesare pe cealaltă, unde își vor continua execuția.

În timp ce abordările menționate anterior se concentrează pe predictibilitate, algoritmul introdus în [105] are ca scop reducerea consumului de energie. De asemenea, obiectivul principal al algoritmului propus în [26] este de a oferi o planificare periodică, non-preemptivă și o valoare a jitter-ului cât mai scăzută pentru transmiterea de mesaje în sisteme cu niveluri mixte de criticalitate, într-un mediu time-triggered.

Jitter-ul unui task este calculat ca fiind diferența dintre separarea maximă și minimă între două job-uri consecutive ale aceluiași task τ_i [106]:

$$Jitter(\tau_i) = \max_{k\geq 1}\{|\, s_{i,k} - s_{i,k+1}|\} - \min_{k\geq 1}\{|\, s_{i,k} - s_{i,k+1}|\} \quad (2\text{-}3)$$

unde:

- $s_{i,k}$ este momentul de start pentru job-ul k al task-ului τ_i.

Cu toate acestea, nici una din metodele menționate anterior nu poate să garanteze o execuție fără jitter într-un sistem cu niveluri mixte de criticalitate, pentru toate task-urile active, indiferent de modul de rulare al sistemului. Acest aspect sintetizează o cerință importantă pentru multe sisteme critice și reprezintă problema pe care această teză își propune să o rezolve.

Cercetarea în cazul metodelor ierarhice este încă în stadiul său preliminar. Astfel, un algoritm ierarhic a fost introdus în [32] pentru planificarea în timp real a task-urilor pe o platformă cu mai multe unități de procesare, cu niveluri mixte de criticalitate. Metoda oferă izolare temporală între task-uri de criticalitate diferită, permițând în același timp redistribuirea de „slack time" între diferite niveluri de criticalitate. Același algoritm a fost implementat și testat pe un sistem de operare standard de timp real (RTOS: Real Time Operating System) [107]. Rezultatele experimentale au arătat că overhead-urile ce țin de RTOS sunt păstrate la niveluri acceptabile, iar sistemul are un comportament robust cu privire la depășirea timpului de execuție pentru cazul cel mai favorabil.

Metodele de planificare time-triggered se mai numesc și algoritmi de planificare offline (sau statici) deoarece planificarea task-urilor pentru fiecare nivel de criticalitate este obținută înainte ca sistemul să își înceapă execuția, oferind predictibilitate. Un dezavantaj al acestor metode este că nu pot planifica în mod eficient task-urile aperiodice (care se repetă la intervale de timp aleatorii) și sporadice (asemănătoare cu cele aperiodice, însă cu o separare minimă între două instanțe consecutive ale aceluiași task), deoarece apariția lor nu este predictibilă [108]. Algoritmii de planificare time-triggered sunt împărțiți în două categorii [31]: table-driven și ciclici.

2.4.1 Algoritmi de planificare table-driven

Tehnicile table-driven calculează și salvează timpii de start ai task-urilor într-un tabel al planificării înainte de începerea rulării sistemului pentru fiecare nivel de criticalitate. În cazul unui sistem cu două niveluri de criticalitate (Hi – criticalitate ridicată, Lo – criticalitate scăzută) vor fi două tabele ale planificării. Astfel, pentru un set cu 2 task-uri de criticalitate Lo, $\{\tau_1, \tau_3\}$ și 3 task-uri de criticalitate Hi, $\{\tau_2, \tau_4, \tau_5\}$ cele două tabele ale planificării sunt prezentate în Tabel 3 și Tabel 4:

Tabel 3. Exemplu de tabel al planificării pentru un algoritm table-driven în modul de criticalitate Lo.

Task	Timp de start (TS)
τ_1	$TS_{1,Lo}$
τ_2	$TS_{2,Lo}$
τ_3	$TS_{3,Lo}$
τ_4	$TS_{4,Lo}$
τ_5	$TS_{5,Lo}$

Tabel 4. Exemplu de tabel al planificării pentru un algoritm table-driven în modul de criticalitate Hi.

Task	Timp de start (TS)
τ_2	$TS_{2,Hi}$
τ_4	$TS_{4,Hi}$
τ_5	$TS_{5,Hi}$

2.4.2 Algoritmi de planificare ciclici

Abordarea ciclică este foarte populară și utilizată frecvent în industrie. Marea majoritate a sistemelor incorporate, dezvoltate în prezent, se bazează pe algoritmi de planificare ciclici. Metodele de planificare ciclice repetă o secvență de frame-uri (sau cicluri minore). Fiecare secvență de frame-uri constă dintr-o serie de instanțe. Durata de execuție a acestei secvențe de frame-uri mai este numită și ciclu major [109]. Aici, punctele de planificare apar la limitele fiecărui frame, deci un task își poate începe execuția doar la începutul unui frame.

Task-urile sunt alocate să ruleze în unul sau mai multe frame-uri, conform tabelelor planificării pentru fiecare mod de rulare al sistemului. Asemenea algoritmilor table-driven, pentru un sistem cu două niveluri de criticalitate (Hi – criticalitate ridicată, Lo – criticalitate scăzută) vor fi două tabele ale planificării. De exemplu, în cazul unui set cu 3 task-uri de criticalitate Lo, $\{\tau_2, \tau_3, \tau_6\}$ și 3 task-uri de criticalitate Hi, $\{\tau_1, \tau_4, \tau_5\}$ cele două tabele ale planificării sunt prezentate în Tabel 5 și Tabel 6:

Tabel 5. Exemplu de tabel al planificării pentru un algoritm ciclic în modul de criticalitate Lo.

Task	Frame
τ_1	$f_{1,Lo}$
τ_2	$f_{2,Lo}$
τ_3	$f_{3,Lo}$
τ_4	$f_{4,Lo}$
τ_5	$f_{5,Lo}$
τ_6	$f_{6,Lo}$

Tabel 6. Exemplu de tabel al planificării pentru un algoritm ciclic în modul de criticalitate Hi.

Task	Frame
τ_1	$f_{1,Hi}$
τ_4	$f_{4,Hi}$
τ_5	$f_{5,Hi}$

2.5 Time-triggered vs. event-driven

Mai jos sunt evidențiate principalele caracteristici ale metodelor time-triggered și event-driven în cazul unui sistem de timp real cu niveluri mixte de criticalitate [110]:

- **Analiza** – această problemă nu poate fi rezolvată în timpul rulării sistemului. Planificabilitatea unui set de task-uri și constrângerile temporale trebuie să fie asigurate în etapa offline pentru fiecare nivel de criticalitate. În cazul abordărilor time-triggered, acest lucru duce la o planificare statică [111], iar pentru metodele event-driven, analiza timpului de răspuns [112] poate să garanteze respectarea tuturor deadline-urilor. Cele două tehnici, time-triggered și event-driven, sunt potrivite pentru a asigura respectarea deadline-urilor sistemului, chiar și în cazul unei încărcări maxime, dar necesită cunoștințe detaliate despre constrângerile de timp pentru toate nivelurile de criticalitate din sistem.
- **Predictibilitate** – metodele time-triggered urmează o planificare determinată static, iar planificarea task-urilor într-o platformă event-driven este construită dinamic, în timpul rulării sistemului, în funcție de momentele de timp la care se iau deciziile ce țin de planificare. Astfel, pentru abordările event-driven este imposibil să se prezică starea sistemului la un moment dat în timp. Cu toate acestea unele MCSs nu necesită predictibilitate la execuția task-urilor, prezența determinismului fiind suficientă pentru a garanta respectarea tuturor deadline-urilor, indiferent de modul de rulare al sistemului.
- **Fezabilitatea** – testarea funcționalității nu diferă foarte mult pentru cele două tipuri de abordări, accentul punându-se pe modul în care sunt verificate constrângerile de timp pentru fiecare nivel de criticalitate. În ambele instanțe este suficient să se testeze fiecare task din sistem folosind timpul de execuție în cazul cel mai defavorabil pentru toate nivelurile de criticalitate. Planificarea tuturor task-urilor din sistem trebuie apoi asigurată utilizând metode formale. Aceste metode există atât pentru abordări time-triggered, cât și event-driven (analiza timpului de răspuns, planificări determinate static, etc.). Testarea în scenarii de încărcare tipice nu este suficientă atunci când trebuie respectate deadline-uri stricte.
- **Extensibilitatea** – se referă la costul adăugării unor task-uri noi într-un MCS existent. Din punctul de vedere al funcționalității, aceste costuri depind în principal de interacțiunea dintre task-urile noi și cele deja existente. Pentru platforme de timp real deadline-urile trebuie să fie respectate chiar și după extinderea numărului de task-uri din sistem. În cazul metodelor time-triggered, planificarea statică va fi recalculată, în timp ce pentru abordările event-driven analiza timpului de răspuns trebuie să fie repetată pentru fiecare nivel de criticalitate.
- **Utilizarea procesorului** – sistemele de timp real cu niveluri mixte de criticalitate trebuie să răspundă la modificările de stare ale mediului înconjurător într-un interval de timp limitat, din această cauză multe platforme utilizează metode event-driven. Acestea sunt flexibile, dar prezintă o utilizare ridicată a procesorului. Abordările time-triggered, în schimb, oferă o flexibilitate mai mică, dar și o utilizare a procesorului mai scăzută.

3 SISTEME DISTRIBUITE

3.1 Sisteme cyber physical

3.1.1 Tipuri de sisteme cyber physical

CPSs încapsulează numeroase sisteme cu tehnologii avansate din domenii precum: automotive, avionică, dispozitive medicale, platforme industriale, etc. Acestea sunt sisteme de control, modelate prin funcții continue și/sau discrete. Comportamentul temporal al unui task este modelat întotdeauna de valori discrete, chiar daca procesul controlat este unul fizic cu un comportament continuu. Efectuarea unor operații de control în buclă, într-o manieră continuă nu este o abordare fezabilă, deci toate operațiile trebuie realizate la intervaluri de timp discrete [113], fiind necesară transformarea parametrilor funcțiilor continue, în parametrii de timp discret. Procesul de transformare a caracteristicilor funcțiilor continue în parametrii de timp discret este explicat în [113]. Deși modelele de task-uri pentru platforme de timp real au abordat inițial o clasă limitată de sisteme, acestea s-au extins în toate domeniile CPSs. Planificarea MASs (Multi-Agent Systems) pe baza unor modele de task-uri pentru sisteme de timp real din literatura de specialitate a fost studiată în [114], ceea ce pregătește terenul pentru extinderea acestor modele și în sistemele robotice.

În cazul în care componentele aplicației prezintă și cerințe de prioritate, în plus față de parametrii temporali, atunci va exista un graf al priorităților asociat setului de task-uri. Un studiu a celor mai utilizate astfel de modele este evidențiat în [115].

Astfel, la un nivel mai înalt al unei aplicații se pot diferenția seturi de task-uri sau meta-task-uri [116], care pot (sau nu) să prezinte cerințe de prioritate sau temporale.

Pentru toate domeniile CPSs planificarea se realizează în funcție de specificitatea fiecărei aplicații sau sistem. În sistemele automotive, majoritatea aplicațiilor sunt periodice, deci scopul planificării este de a defini timpul de start pentru fiecare task. Metoda de planificare trebuie să ia în considerare toate constrângerile arhitecturale, precum sistemul de comunicație sau sistemul de operare utilizat. Dezvoltarea unor politici de planificare pentru întregul sistem este o provocare, în special când vine vorba de eficiență și scalabilitate [117]. Cu toate acestea, [118] introduce o arhitectură modulară pentru planificarea sistemelor distribuite de tip time-triggered (cu execuție periodică). Spre deosebire de alte abordări, tehnica propusă respectă cerințele specifice platformelor automotive la nivel de sistem și este complet scalabilă. Aplicabilitatea acestei abordări este evidențiată în aceeași lucrare prin planificarea simultană a mai multor funcții de control distribuite.

Un model de task-uri paralele, de timp real, a fost propus în [119] care permite variația numărului de task-uri executate în paralel, conform atributelor fizice ale sistemului. Astfel, un vehicul poate să modeleze în paralel aspecte ale mediului fizic pentru a reacționa la acestea în timp real. Algoritmii de planificare preemptivi cu prioritate fixă sunt cei mai des utilizați în domeniul automotive (OSEK, AUTOSAR) [120].

Sistemele de tip IMA (Integrated Modular Avionics) sunt alcătuite, conform standardului ARINC-653 [121], din unul sau mai multe module, conectate între ele. Modulele sunt platforme hardware formate, la rândul lor, în jurul unuia sau mai multe procesoare. Acestea asigură izolarea temporală și spațială pentru execuția independentă a aplicațiilor, denumite partiții. Fiecare astfel de aplicație este compusă

din procese. Procesele care aparțin aceleiași partiții sunt planificate de un mecanism de planificare local, iar partițiile alocate aceluiași procesor sunt planificate folosind un algoritm de planificare global. În [122] este descrisă o tehnică compozițională pentru planificarea automatizată a acestor aplicații și procese. Accentul se pune pe metode de planificare a aplicațiilor prin intermediul interfețelor acestora, luând în considerare overhead-urile generate de preempțiuni.

Un exemplu de model de sistem IMA este SP-RTS (Strongly Partitioned Real-Time System), care se ocupă cu planificarea partițiilor (aplicațiilor) și a canalelor de comunicare. Pentru planificarea acestor aplicații și canale de comunicare a fost introdus un algoritm în [123]. Ideea metodei este de a folosi două niveluri ierarhice care activează partiții (sau canale) în funcție de un mecanism de planificare ciclic, apoi alocă task-uri utilizând o tehnică de planificare cu prioritate fixă.

Abordarea tradițională pentru planificarea în sisteme de timp real avionice este de a folosi un algoritm de planificare static care eșantionează operațiile de zbor la intervaluri de timp și frecvențe fixe de-a lungul unui zbor. Planificarea condiționată (dinamică) înlocuiește cea statică, pentru a oferi siguranță prin adaptarea planificării în funcție de modificarea cerințelor de rulare [124]. Algoritmii condiționați permit utilizarea eficientă a resurselor deoarece aceștia iau în considerare relațiile situaționale între operatorii de zbor.

Dispozitivele medicale sunt sisteme de timp real care prezintă cerințe de siguranță și securitate. Acestea pot varia de la sisteme incorporate reactive de timp real, precum stimulatoare cardiace, până la distribuitoare de medicație. Un astfel de exemplu este planificarea unui ventilator medical adaptabil [125], utilizând un algoritm (m,k)-firm, pentru a permite ajustarea de la distanță de către clinician a ratei de asistare a dispozitivului. Majoritatea dispozitivelor sunt conectate la rețea, mai ales în unitățile de terapie intensivă din spitale, astfel încât condițiile pacienților detectate de senzori să poată fi monitorizate în timp real [126].

Platformele industriale sunt alcătuite din elemente autonome și colaborative, precum și din subsisteme conectate în funcție de contextul fiecărui nivel de producție, de la procesarea materialelor prime, până la rețele de producție și logistică. Ideea utilizării unei infrastructuri pentru sisteme industriale ce combină planificarea statică cu cea dinamică a fost abordată în [127]. Sistemele industriale cyber physical au început să adopte tehnologii WSANs (Wireless Sensor-Actuator Networks), însă în cazul acestora apar provocări când vine vorba de condițiile dinamice wireless din fabrici. Astfel, o serie de algoritmi de planificare au fost descriși pentru WSANs în contextul sistemelor industriale de control [128-130]. Modularizarea sistemelor automatizate din producție necesită transformarea structurii prezente, orientate pe funcții de control, în module funcționale cu structuri orientate pe task-uri.

3.1.2 Caracteristicile sistemelor cyber physical

Chiar dacă CPSs includ o varietate mare de sisteme, foarte diferite din punct de vedere arhitectural și funcțional, totuși pot fi identificate o serie de atribute comune [65]. Aceste caracteristici au o anumită influență asupra algoritmului de planificare:

- **Eterogenitate**

În contextul CPSs, eterogenitatea se referă la specificații hardware variate, precum și diferite cerințe ce țin de aplicație sau de consumul de putere. Toate aceste aspecte pot fi modelate prin diverși parametri, lucru care se reflectă din teoria planificării în MCSs.

- **Administrare eficientă a puterii**

Administrarea eficientă a puterii trebuie avută în vedere pentru că numeroase componente folosesc surse de alimentare sub formă de baterii în CPSs, ceea ce rezultă în dezvoltarea unor algoritmi de planificare ce țin cont de această problemă, atât pentru sisteme clasice de timp real, cât și pentru MCSs [65, 131, 132].

- **Dinamism și adaptabilitate**

Datorită interacțiunii directe cu mediul înconjurător (fizic), care este de multe ori dinamic și imprevizibil, sistemul trebuie să facă față unor modificări în timp real [23, 133].

- **Robustețe**

Deoarece mediul în care operează CPSs prezintă condiții incerte când vine vorba de comportamentul în timpul rulării sistemului (indisponibilitatea rețelei, defecțiuni hardware, etc.), este de preferat ca aplicațiile critice să nu fie afectate de aceste defecțiuni sau de overload-ul computațional cauzat de alte aplicații. Acest aspect se reflectă în necesitatea de a utiliza mecanisme hypervisor sau de monitorizare.

- **Redundanță**

Pentru a evita producerea unui SPOF (Single Point of Failure), este esențială atât redundanța hardware, cât și cea computațională. Astfel, politicile de planificare dinamice sunt necesare datorită capacității de a se adapta la diferite sarcini computaționale și de a replanifica aplicațiile noi venite.

- **Distribuție**

CPSs sunt alcătuite din componente variate care comunică și interacționează între ele (aplicații robotice sau rețele wireless de senzori). În acest sens, trebuie considerați algoritmi de planificare pentru sisteme cu mai multe unități de procesare care țin cont de locația componentelor și de mediul de comunicare dintre acestea. O abordare ierarhică, în care una sau mai multe entități coordonează subsisteme diferite poate fi necesară.

- **Scalabilitate**

Atunci când sunt utilizate sisteme cu mai multe unități de procesare, scalabilitatea acestora în termeni de model, componente hardware, analiză, codare și testare este foarte importantă. Din această cauză trebuie asigurată scalabilitatea modelelor de task-uri, a mecanismelor de planificare, precum și a analizei planificării.

- **Securitate**

Asemenea tuturor platformelor computaționale, CPSs pot fi supuse unor atacuri malițioase. Mai mult, un sistem complex care conține funcționalități de criticalitate diferită este mai greu de protejat. Trebuie acordată importanță sporită, mai ales aplicațiilor critice, deci o implicare directă în acest sens asupra politicii de planificare este de a asigura izolarea între aplicațiile critice și cele care nu sunt critice în MCSs.

- **Izolare**

Într-un sistem complex, care conține funcționalități diferite, componentele critice trebuie izolate de cele care nu sunt critice atât din punctul de vedere al comportamentului temporal, cât și a utilizării resurselor.

Dezvoltarea unor mecanisme de planificare care să incorporeze aceste atribute este o adevărată provocare datorită aplicabilității vaste a CPSs în diferite domenii, precum și lipsei unor benchmark-uri și metrici comune, potrivite pentru toți algoritmii prezentați anterior. O serie de niveluri de conformitate, introduse și în [17], sunt prezentare în continuare:

Tabel 7. Niveluri de conformitate.

Atribut	Nivel	Descriere nivel
Eterogenitate	La nivel de task	Gestionează diferite tipuri de seturi de task-uri
	La nivel de dispozitiv	Gestionează diferite arhitecturi
Administrarea eficientă a puterii	-	Nu ia în considerare administrarea puterii
	Scăzut	Ia în considerare doar consumul de putere statică
	Mediu	Ia în considerare doar consumul de putere dinamică
	Ridicat	Ia în considerare atât consumul de putere statică, cât și dinamică
Dinamism și adaptabilitate	Scăzut	Nu acceptă încărcarea dinamică a task-urilor
	Mediu	Acceptă încărcarea dinamică a task-urilor, dar în regim limitat
	Ridicat	Seturile de task-uri pot fi încărcate dinamic în timpul rulării
Robustețe	Scăzut	Nu gestionează overload
	Mediu	Gestionează overload doar în cazul nivelurilor de criticalitate ridicată
	Ridicat	Gestionează overload atât în cazul nivelurilor de criticalitate scăzută, cât și a celor de criticalitate ridicată
Distribuție	Da/Nu	Algoritmi pentru sisteme distribuite/Alți algoritmi
Scalabilitate	Da/Nu	Scalabil cu privire la numărul de niveluri de criticalitate/Nu este scalabil
Securitate și izolare	Scăzut	Doar izolare temporală între nivelurile de criticalitate
	Mediu	Izolare temporală și spațială doar în cazul nivelurile de criticalitate ridicată
	Ridicat	Izolare spațială și temporală între diferite niveluri de criticalitate

Tabel 8 prezintă o evaluare a algoritmilor de planificare pentru sisteme cu o singură unitate de procesare pe baza nivelurilor de conformitate descrise în Tabel 7 [17].

Tabel 8. Niveluri de conformitate pentru diferiți algoritmi în sisteme cu o singură unitate de procesare.

Algoritm de planificare	Administrarea eficientă a puterii	Dinamism și adaptabilitate	Robus-tețe	Distri-buție	Scalabilitate	Securitate
FTP-SMC	-	Scăzut	Scăzut	Nu	Da	Scăzut
FTP-AMC	-	Scăzut	Mediu	Nu	Da	Scăzut
FTP-preemptive	-	Scăzut	Mediu	Nu	Da	Scăzut
ZSS	-	Scăzut	Mediu	Nu	Da	Scăzut
FJP-OCBP	Ridicat	Scăzut	Scăzut	Nu	Da	Scăzut
EDF-VD	Ridicat	Ridicat	Mediu	Nu	Da	Scăzut
EDF-DB	-	Ridicat	Mediu	Nu	Da	Scăzut
EDF-AD	-	Ridicat	Mediu	Nu	Nespecificat	Scăzut
EDF-AD-E	-	Ridicat	Mediu	Nu	Nespecificat	Scăzut
CBEDF	-	Ridicat	Ridicat	Nu	Nu	Scăzut
PLRS	-	Mediu	Scăzut	Nu	Da	Scăzut

Nivelul fiecărui atribut definit în acest capitol, în cazul algoritmilor de planificare pentru sisteme cu mai multe unități de procesare, este evidențiat în Tabel 9, pe baza descrierilor prezentate în Tabel 7 [17]. Această analiză este utilă la incorporarea algoritmilor de planificare pentru MCSs în domeniile CPSs sau de a folosi aceste tehnici ca bază pentru dezvoltarea unor modele noi, mai fezabile.

Tabel 9. Niveluri de conformitate pentru diferiți algoritmi în sisteme cu mai multe unități de procesare.

Algoritm de planificare	**Etero-genitate**	**Adminis-trarea eficientă a puterii**	**Dinamism și adapta-bilitate**	**Robus-tețe**	**Distri-buție**	**Scalabili-tate**	**Securi-tate**
MC-PARTITION	La nivel de task	-	Mediu	Mediu	Nu	Nespecificat	Scăzut
MC-PARTITION-UT-0.75	La nivel de task	-	Mediu	Mediu	Nu	Nespecificat	Scăzut
MC-PARTITION-UT-1	La nivel de task	-	Mediu	Mediu	Nu	Nespecificat	Scăzut
MC-PARTITION-UT-INC	La nivel de task	-	Mediu	Mediu	Nu	Nespecificat	Scăzut
TTS-MC	La nivel de task	-	Mediu	Mediu	Nu	Nespecificat	Scăzut
Event Scheduler in Multi-Core	La nivel de task	-	Mediu	Mediu	Nu	Nespecificat	Scăzut
MC-IS-Server	La nivel de task	-	Mediu	Mediu	Nu	Da	Scăzut
TG-PEDF	La nivel de task	-	Mediu	Ridicat	Nu	Da	Mediu
NPS-F-IMA	La nivel de task	-	Mediu	Ridicat	Nu	Nespecificat	Ridicat
P-EKB	La nivel de task	-	Mediu	Mediu	Nu	Nespecificat	Scăzut
P-EDF-VD	La nivel de task	-	Mediu	Mediu	Nu	Da	Scăzut
FTTS	La nivel de task	-	Scăzut	Mediu	Nu	Da	Scăzut
EM3	La nivel de task	Ridicat	Mediu	Mediu	Nu	Nespecificat	Scăzut
IM3	La nivel de task	Ridicat	Mediu	Ridicat	Nu	Nespecificat	Ridicat
Abordarea lui Ali și Kim	La nivel de task	-	Mediu	Mediu	Nu	Da	Scăzut
NPS-F-MC	La nivel de task	-	Ridicat	Ridicat	Nu	Nespecificat	Ridicat
SP-EKB	La nivel de task	-	Ridicat	Mediu	Nu	Nespecificat	Scăzut
MxC-RUN	La nivel de task	-	Ridicat	Mediu	Nu	Nespecificat	Scăzut
LPDPM-MC	La nivel de task	Scăzut	Scăzut	Ridicat	Nu	Nespecificat	Scăzut
GLOBAL	La nivel de task	-	Ridicat	Mediu	Nu	Nespecificat	Scăzut
GLOBAL-PRAGMATIC	La nivel de task	-	Ridicat	Mediu	Nu	Nespecificat	Scăzut
MC-Fluid	La nivel de task	-	Ridicat	Mediu	Nu	Nespecificat	Scăzut
MC-IS-Fluid	La nivel de task	-	Ridicat	Mediu	Nu	Da	Scăzut
F_MHEFT	La nivel de dispozitiv	-	Scăzut	Scăzut	Da	Da	Nespecificat
D_MHEFT	La nivel de dispozitiv	-	Scăzut	Mediu	Da	Da	Nespecificat

FDS_MIMF	La nivel de dispozitiv	-	Ridicat	Mediu	Da	Da	Nespecificat
ADS_MIMF	La nivel de dispozitiv	-	Ridicat	Mediu	Da	Da	Nespecificat

Algoritmii de planificare distribuiți prezintă cel mai mare potențial pentru CPSs datorită scalabilității, dinamismului și capacității de adaptare, care sunt aspectele principale ale unui mediu aflat într-o continuă schimbare. Tehnicile de planificare F_MHEFT și D_MHEFT [94] au fost introduse inițial pentru platforme incorporate distribuite eterogene dar în [94] sunt, de asemenea, prezentate din perspectiva sistemelor automotive. Cu toate acestea ambele metode pot fi integrate în cadrul altor domenii CPS, precum sisteme industriale sau Internet of Things. Asemănător, algoritmii FDS_MIMF și ADS_MIMF [95] au fost dezvoltați în principal pentru automotive CPSs cu scopul de a satisface anumite cerințe specifice domeniului. FDS_MIMF poate fi aplicat și în avionică și sisteme industriale, iar ADS_MIMF este potrivit pentru domenii cu restricții temporale și cerințe de securitate foarte stricte (avionică și dispozitive medicale).

3.1.3 Integrarea sistemelor cu niveluri mixte de criticalitate în sistemele cyber physical

3.1.3.1 Provocări ale sistemelor cu niveluri mixte de criticalitate

Deoarece MCSs sunt într-o continuă dezvoltare, există o serie de provocări care pot să apară:

- **Izolarea** – este unul dintre cele mai importante aspecte în MCSs. Până în momentul de față, studiile au abordat soluții centrate pe sistem sau aplicație. Pentru a rezolva aceste provocări, s-au adus numeroase contribuții în diferite domenii. Un astfel de exemplu este extensia LITMUS [134] pentru nucleul Linux, dezvoltată cu scopul studiului și evaluării algoritmilor de planificare. O altă cerință importantă în MCSs este izolarea temporală pentru a evita interferența între task-urile de criticalitate scăzută și cele de criticalitate ridicată. LITMUS asigură izolarea temporală între componente, pentru ca cerințele de timp să fie validate independent. Astfel, aceste soluții pot fi extinse pentru a include alte sisteme integrate sau diverse sisteme de operare de timp real.
- **Modele unitare** – o altă problemă provine din lipsa unor modele unitare în domeniile CPSs, atât la nivel de task, cât și la nivel de planificare. Unele domenii nu au o abordare standardizată pentru planificare, precum domeniul medical [135] sau Internet of Things [136]. Acest lucru poate să conducă la probleme când vine vorba de integrarea MCSs în CPSs, deoarece o abordare unitară necesită algoritmi de planificare generici, bine definiți, care pot fi adaptați în funcție de aplicație.
- **Planificarea task-urilor în contextul izolării nivelurilor de criticalitate și a eficienței** – conceptul de bază pentru dezvoltarea unui CPS este de a demonstra că există suficientă independență între nivelurile de criticalitate. Planificarea task-urilor în CPSs este o provocare din cauza câtorva factori, precum: atribuirea priorității, considerarea atât a nivelului de criticalitate, cât și a specificațiilor de timp, asigurarea izolării între

diferitele niveluri de criticalitate, partajarea și utilizarea eficientă a resurselor, etc.

- **Comunicarea în timp real** – într-un sistem complex cu componente de timp real și componente care nu sunt de timp real, comunicarea în timp real trebuie asigurată între toate componentele. Pentru a avea un sistem distribuit de timp real, toate componentele trebuie să respecte acest model. În consecință, existența unui astfel de suport de comunicare este obligatorie.
- **Partajarea resurselor** – acest lucru este mai dificil de atins în contextul izolării în timp și a funcționalităților. Task-urile de criticalitate ridicată au prioritate mai mare decât cele de criticalitate scăzută când vine vorba de alocarea resurselor.

Alte probleme care intervin din cauza specificității CPSs și care ar putea împiedica procesul de integrare a conceptului de niveluri mixte de criticalitate în CPSs, includ:

- **Nevoia de eficiență energetică –** acest lucru este crucial în CPSs colaborative, ce conțin de cele mai multe ori componente mobile, care nu sunt alimentate direct de la rețeaua electrică. Astfel, trebuie dezvoltați algoritmi de planificare speciali care iau în considerare acest aspect.
- **Hardware eterogen –** deoarece sistemele colaborative pot avea arhitecturi și configurații diferite, iar în cadrul unui sistem colaborativ pot fi diverse componente cu un suport hardware variat, o abordare unitară în implementarea MCSs este dificil de atins.
- **Protocoale de comunicație variate –** din cauza utilizării unor componente hardware diferite și a unor arhitecturi structurate în mod ierarhic, diverse protocoale de comunicație pot fi folosite între componentele platformei (sau nivelurile arhitecturii), mărind complexitatea sistemului.

Deoarece conceptul de CPS este vast, acoperind multiple domenii, este importantă sublinierea provocărilor care apar o dată cu integrarea modelului cu niveluri mixte de criticalitate în CPSs. Două aspecte care trebuie luate în considerare sunt evidențiate în continuare:

- **Specificitatea domeniului** – în unele domenii lipsește standardizarea, care ar putea fi aplicată la diferite niveluri:
  - La nivel de task – modele generice pentru diferite niveluri ale unei aplicații, compatibile cu diverși algoritmi de planificare.
  - La nivel de planificare – metode de planificare standardizate care pot fi folosite în toate domeniile CPSs.

Folosirea unor standarde bine definite oferă consistență, calitate și mărește productivitatea. Acest lucru implică un set de reguli care trebuie să fie ușor de înțeles, urmate în mod constant și îmbunătățite în permanență.

- **Generalitate** – deși algoritmii de planificare generici există la nivel teoretic, aplicabilitatea lor practică nu a fost abordată, majoritatea implementărilor fiind specifice fiecărei aplicații sau sistem. În consecință, soluții pentru aceeași problemă pot varia în funcție de domeniu. Chiar dacă aplicarea unor rezolvări specific adaptate oferă determinism, predictibilitate și performanță ridicată în anumite scenarii, soluțiile generale aduc o mai mare flexibilitate, scalabilitate și interconectivitate între diferite dispozitive.

Aceste probleme pot fi depășite folosind o abordare standardizată, aplicabilă în toate domeniile CPSs. Cu toate acestea, caracteristicile comune de funcționare și

design trebuie luate în considerare pentru fiecare arie de dezvoltare. În plus, utilizarea unor algoritmi de planificare generici este mult mai avantajoasă decât metodele orientate pe aplicație, atât din punctul de vedere al flexibilității, cât și al costului.

Soluționarea acestor provocări va duce la o integrare eficientă a MCSs în CPSs, cu avantaje majore pe termen lung.

3.1.3.2 Avantajele integrării sistemelor cu niveluri mixte de criticalitate în sistemele cyber physical

În ciuda numeroaselor provocări, integrarea modelului cu niveluri mixte de criticalitate în CPSs, aduce o serie de avantaje:

- **Funcționalități de timp real** – CPSs conțin funcționalități precum adaptabilitate și securitate, ceea ce implică respectarea unor constrângeri de timp stricte când vine vorba de modificări ale mediului, procesarea datelor, partajarea informațiilor și executarea anumitor acțiuni. Deși, inițial modelul cu niveluri mixte de criticalitate a fost dezvoltat pentru sisteme de timp real, acesta poate fi integrat și pe platforme cu cerințe de timp real și cerințe care nu sunt de timp real, atunci când funcționalități și componente diferite, critice sau nu, împart același hardware [32].
- **Funcționalități multiple pe aceeași platformă** – implementarea unor funcționalități multiple pe aceeași platformă aduce multe avantaje. În consecință, unele componente vor fi mai critice decât altele, ceea ce face ca modelul cu niveluri mixte de criticalitate să fie luat in considerare tot mai des în CPSs.
- **Planificare ierarhică** – marea majoritatea a algoritmilor de planificare sunt dezvoltați la nivelul de bază al unei aplicații (de exemplu la nivel de task), dar în contextul sistemelor distribuite complexe, unde este necesar ca aplicații de la furnizori diferiți să funcționeze simultan, această abordare devine ineficientă. Adițional, interconectarea obiectivelor ce țin de planificarea sistemului între diferite niveluri computaționale ale unui sistem complex asigură transparență și ajută la atingerea cerințelor de timp. Adoptarea modelelor unitare de abstractizare pentru diferite niveluri ale unei aplicații (de exemplu task, meta-task, sau la nivel de sistem) permite interconectarea CPSs, indiferent de domeniul de aplicabilitate, aducând astfel lucrurile mai aproape de o abordare standardizată.
- **Abordări standardizate** – în prezent, fiecare domeniu al CPSs are proprii algoritmi, dezvoltați în funcție de anumite cerințe și de comportamentul în timp real. Acest lucru creează inconsistență la aplicarea aceleași metode de planificare unor domenii diferite. Astfel, trebuie să se ajungă la o soluție comună, prin integrarea conceptului de niveluri mixte de criticalitate, pentru o mai bună interconectare a componentelor, indiferent de nivelul de criticalitate și pentru a pune bazele unor oportunități viitoare de dezvoltare.
- **Izolarea temporală și a resurselor** – ca urmare a cercetărilor întreprinse în domeniul MCSs din ultimul deceniu, care țin mai ales de planificare, au fost prezentate o serie de rezultate importante. Una din consecințele acestor cercetări este faptul că sistemele critice nu mai sunt limitate la un singur tip de unități de procesare. Acest lucru se reflectă în varietatea de sisteme eterogene dezvoltate, cu performanțe computaționale și de putere diferite. Mecanismele pentru izolare temporală și a resurselor permit utilizarea unor unități de procesare

avansate, cu capabilități computaționale superioare, dar care oferă mai puțină predictibilitate în sistemele critice, unde securitatea este esențială [135].

- **Adaptabilitate** – utilizarea unui model cu valori diferite a parametrilor pentru fiecare mod de rulare permite realizarea de predicții cu privire la evoluția parametrilor care țin de performanța sistemului (de exemplu energia utilizată la rulare, memoria utilizată, lățimea de bandă pentru comunicare, etc.) [135]. Acest aspect este de interes într-un sistem cyber physical, în care datorită interacțiunii dintre mediul înconjurător și sistem, se impune adaptarea dinamică a platformei la diferite condiții de rulare, în funcție de variațiile mediului. În acest sens capacitatea de a modela și testa sistemul offline este un progres important.
- **Sistem robust** – tehnicile care folosesc hypervisor și middleware standard distribuit reprezintă soluții generale pentru a impune robustețe în ceea ce privește comportamentul sistemului la rulare.

Aceste avantaje oferă posibilitatea unor cercetări viitoare în domeniu.

3.1.3.3 Perspective de cercetare

Pe lângă avantajele, preponderent de ordin practic, menționate anterior relativ la integrarea MCSs în CPSs, se conturează noi perspective de cercetare:

- Planificarea în sisteme IoT cu niveluri mixte de criticalitate – sistemele IoT evoluează rapid înspre sisteme IoT de timp real [137], în care task-uri de criticalitate diferită, cu constrângeri de timp diferite, coexistă. Astfel, implementarea unei abordări ierarhice pentru planificare, ce poate fi extinsă pentru a include diferite niveluri de planificare IoT, precum planificarea la nivel de dispozitiv, planificarea Fog și planificarea la nivel de servicii cloud, devine o provocare considerabilă.
- Fog computing [138] este o paradigmă nouă, în cadrul căreia un număr mare de unități de procesare eterogene și descentralizate, comunică și cooperează între ele. Ca și consecință, în afară de nevoia de standardizare, integrarea MCSs în Fog computing trebuie să îndeplinească niște cerințe adiționale: sincronizare, securitate și programabilitate. Acest lucru se concretizează într-o nouă direcție de cercetare cu potențial ridicat, și anume conceptul de planificare Fog.
- Planificarea task-urilor cu niveluri mixte de criticalitate în MAS (Multi-Agent Systems) – un set similar de provocări cu cele prezentate anterior au fost scoase în evidență de Calvaresi et al. [11], la integrarea conceptului de MCS în alte domenii CPS: lipsa standardizării, cerințe de timp real (comunicare, partajarea informației, adaptarea la schimbările din mediu, etc.), precum și lipsa unui model de task-uri care să ia în considerare comportamentul temporal. Astfel, soluționarea acestor provocări va avea un impact major asupra dezvoltării mai multor zone de cercetare, precum MAS sau IoT.
- Implementările MCS în CPSs au potențialul de a aduce îmbunătățiri majore ce țin de securitate prin integrarea conceptului de izolare temporală și spațială.

Luând în considerare aceste zone de cercetare, precum și algoritmii de planificare clasificați în Capitolul 2 se pot face următoarele observații:

- Sisteme IoT – Utilizarea conceptului de niveluri mixte de criticalitate în IoT va îmbunătăți fiabilitatea, siguranța, scalabilitatea și eficiența

transmisiei sistemelor prin extinderea algoritmilor folosiți la planificarea mesajelor, pentru a gestiona niveluri de criticalitate diferite și cerințe de timp real. Tehnologiile web existente sunt ineficiente la interfațarea cu sisteme IoT de timp real când vine vorba de planificarea mesajelor. Din această cauză, o abordare eterogenă distribuită, precum D_MHEFT sau F_MHEFT prezintă potențial, oferind flexibilitate și adaptabilitate pentru a coordona componente multiple într-un mediu dinamic.

- Planificarea Fog – același concept poate fi aplicat și la nivel de Fog computing. În acest caz, se pot folosi algoritmi de planificare, atât pentru sisteme cu o singură unitate de procesare, cât și pentru sisteme cu mai multe unități de procesare, în funcție de complexitatea hardware-ului. Din punctul de vedere al platformelor cu o singură unitate de procesare, algoritmii care au la bază modelul EDF, precum EDF-VD sau CBEDF, prezintă potențial mare deoarece sunt dinamici și eficienți din punct de vedere energetic. Câteva metode cunoscute de planificare pentru sisteme cu mai multe unități de procesare, aplicabile și în Fog computing sunt: P-EDF-VD, P-EKB pentru planificarea partiționată, MxC-RUN, MC-Fluid pentru planificarea globală și NPS-F-MC, SP-EKB pentru planificarea cluster/semi-partiționată.
- MAS – un design ierarhic și descentralizat este potrivit, integrând mai multe clase de algoritmi de planificare pentru fiecare nivel. De exemplu, în cazul unei flote de roboți: la nivel de componentă se pot folosi algoritmi de planificare pentru platforme cu o singură unitate de procesare, precum EDF-VD și EDF-DB, la nivel de agent, metode partiționate, precum TTS-MC sau MC-PARTITION-UT-INC, iar la nivel de sistem o abordare distribuită.

3.2 Internet of Things

3.2.1 Arhitecturi bazate pe Fog în IoT

IoT a fost aplicat pe scară largă în numeroase domenii, precum cel medical [139] sau sisteme industriale [140, 141]. Astfel, diferite tipuri de arhitecturi fizice pentru IoT au fost propuse și descrise în literatura de specialitate: arhitecturi clasice pe mai multe niveluri [10, 142], arhitecturi bazate pe „IoT gateway” [143], sau arhitecturi bazate pe Fog [144, 145]. Dintre acestea, arhitectura bazată pe Fog este cea mai potrivită pentru aplicațiile de timp real distribuite complexe. Această infrastructură permite ca execuția și stocarea să se realizeze în locații diferite (Cloudlets) [146].

3.2.1.1 Fog vs. Cloud

Arhitecturile bazate pe Fog sunt alcătuite din trei niveluri [147]: Cloud (unde datele sunt stocate în centre de date și livrate ca servicii utilizatorilor prin intermediul Internetului), Fog (în care procesarea și stocarea datelor se realizează local, cât mai aproape de utilizatorii finali, pentru a elimina întârzierile din rețea) și Edge (este o rețea de dispozitive eterogene și distribuite). Nivelul Fog poate să ofere suport pentru funcțiile de control „time-sensitive” a nodurilor de la nivelul Edge [145]. În prezent nivelul Cloud nu poate să ofere aceste servicii, din cauza mai multor factori. Unul dintre acești factori este lipsa unui protocol de comunicare în timp real între Cloud și

Edge, comunicarea fiind realizată utilizând protocoale de Internet, care în mod obișnuit nu sunt de timp real. Figura 4 prezintă arhitecturile de bază a celor două niveluri, Fog și Cloud:

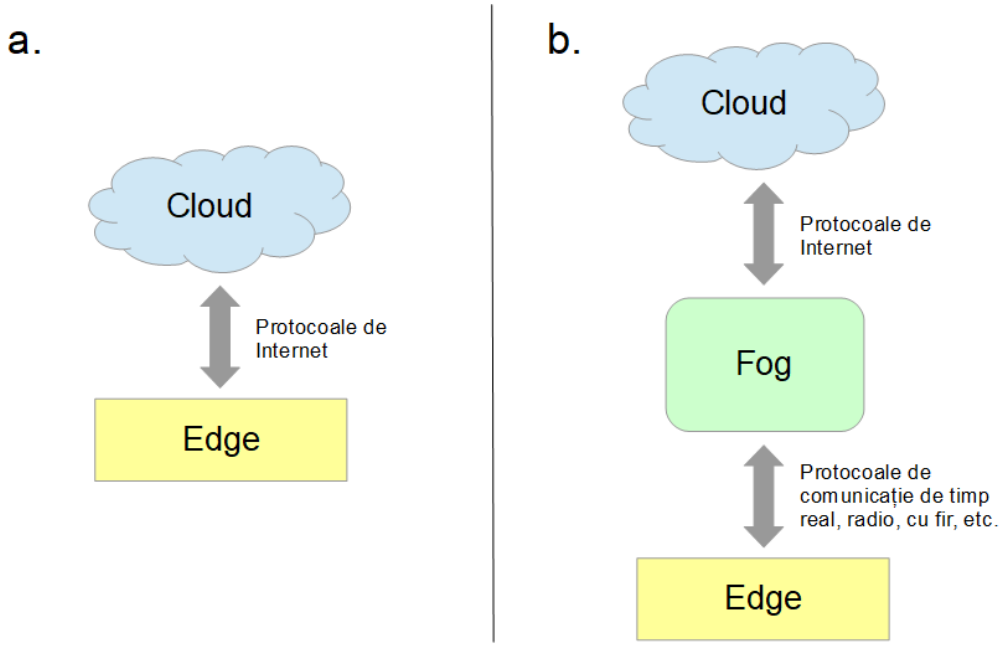


Figura 4. Arhitectura de bază: a. Cloud și b. Fog.

De asemenea, o comparație între nivelurile Fog și Cloud este evidențiată în Tabel 10 [148]:

Tabel 10. Comparație între nivelurile Cloud și Fog.

Cerințe	**Nivelul Cloud**	**Nivelul Fog**
Latență	Ridicată	Scăzută
Hardware	Putere de stocare și procesare scalabilă	Putere de procesare și stocare limitată
Locația nodurilor server	În cadrul Internetului	La periferia rețelei locale
Distanța dintre client și server	Dispozitive intermediare multiple	Nu există dispozitive intermediare
Mediul de lucru	Depozite cu sisteme de ventilație	Exterior (străzi, grădini, etc.) sau interior (restaurante, spitale, etc.)
Măsuri de securitate	Bine definite	Greu de definit
Atacuri la date	Probabilitate scăzută	Probabilitate ridicată
Infrastructură	Centralizată	Distribuită
Perceperea locației	Nu	Da
Suport pentru mobilitate	Mobilitate limitată	Suportă mobilitatea
Procesarea în timp real	Nu	Da

3.2.1.2 Caracteristicile arhitecturilor bazate pe Fog

Această secțiune rezumă principalele caracteristici ale unei arhitecturi bazate pe Fog [149-152]:

- **Eterogenitatea** – nodurile Fog sau dispozitivele IoT sunt proiectate de către producători diferiți, în consecință prezintă diverse resurse de procesare, stocare și rețea și trebuie amplasate în funcție de platformă. Infrastructurile Fog au abilitatea de a lucra pe platforme diferite.
- **Perceperea locației și latența scăzută** – conceptul de Fog a aparut ca urmare a necesității de a furniza suport pentru dispozitivele de la nivelul Edge care oferă servicii complexe. Nodurile Fog pot să fie amplasate în locații diferite, dar cu cât sunt mai aproape de nodurile IoT, cu atât oferă o latență mai scăzută la procesarea datelor.
- **Distribuția geografică** – spre deosebire de infrastructura Cloud care este centralizată, serviciile și aplicațiile de la nivelul Fog necesită implementări distribuite pe scară largă.
- **Numărul foarte mare de noduri datorită distribuției geografice** – acest aspect este evidențiat în special în infrastructurile Smart Grid [153, 154], dar și în rețelele de senzori.
- **Scalabilitatea** – scopul arhitecturilor Fog este de a reduce traficul de date între Cloud și Edge, prin procesarea datelor cât mai aproape de Edge, ceea ce rezolvă problema scalabilității, care apare o dată cu creșterea numărului de noduri Edge [155].
- **Suport pentru mobilitate** – în aplicațiile Fog este importantă comunicarea directă cu dispozitivele mobile, prin intermediul anumitor tehnici, pentru a oferi metode de accesare rapidă și analiză a datelor și a serviciilor de la nivelul nodurilor IoT [146] (de exemplu protocolul Locator/ID Separation (LISP)) [156]. Acesta separă identitatea host-ului de cea a locației.
- **Procesarea în timp real și administrarea eficientă a resurselor de stocare** – prin extinderea funcționalităților Cloud către limitele rețelei, nivelul Fog permite procesarea datelor recepționate de la dispozitivele IoT în timp real și generarea de acțiuni instantanee. În acest mod se evită transmiterea unui volum mare de date către centrele de procesare și analiză din Cloud, întrucât majoritatea informațiilor sunt sortate și indexate local, în Fog.
- **Interoperabilitatea** – componentele Fog pot lucra cu domenii diferite, folosind servicii de la diverși furnizori [157].
- **Suport pentru analiza online și interacțiunea cu infrastructura Cloud** – nivelul Fog este situat între Cloud și dispozitivele IoT cu scopul de a elimina congestiile din trafic prin recepționarea și procesarea locală a datelor, în apropierea nodurilor Edge.

Caracteristicile prezentare anterior sunt ilustrate în Figura 5:

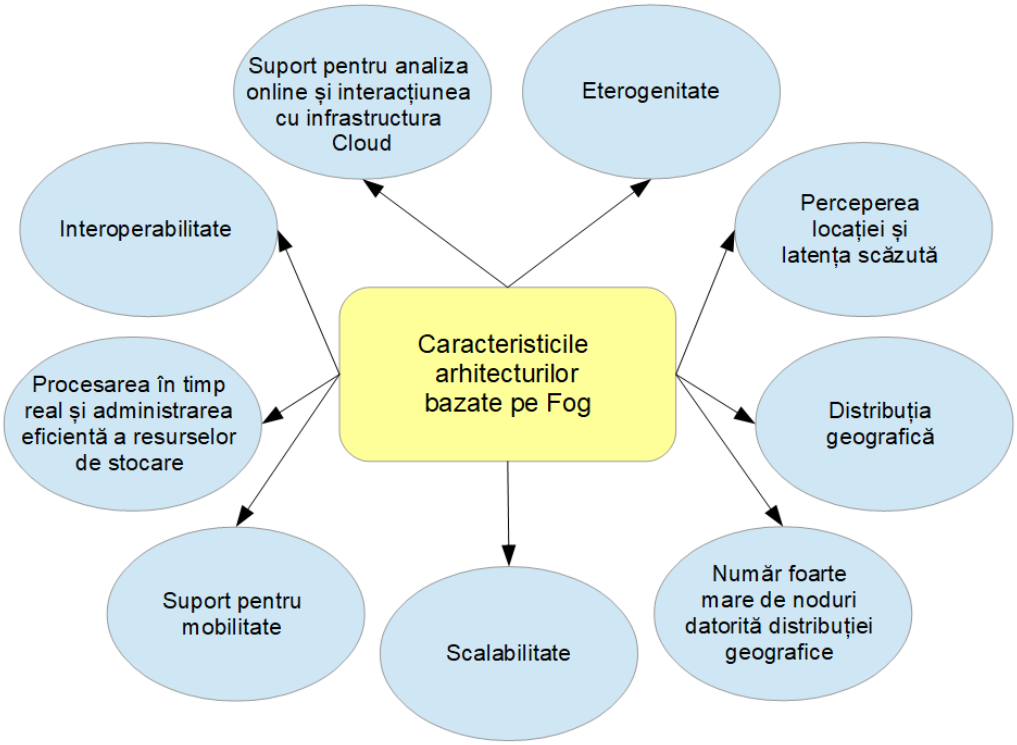


Figura 5. Caracteristicile arhitecturilor bazate pe Fog.

În cadrul nivelului Fog, nodurile sunt eterogene și distribuite [158]. Acestea pot îndeplini roluri importante, precum:

- **Furnizarea de servicii „latency-sensitive”** – oferirea unor servicii cu cerințe de timp real care sunt dificil de furnizat, sau chiar imposibil, la nivel de Cloud, întrucât tehnologia Cloud nu vizează aplicațiile hard de timp real [159].
- **Crearea unei căi de comunicare** – asigurarea comunicării între rețelele de timp real și rețelele care nu sunt de timp real.
- **Planificarea în timp real și administrarea resurselor** – planificarea locală a aplicațiilor la nivel de Fog, sau planificarea globală a rețelelor IoT de dispozitive distribuite la nivel de Edge.
- **Izolarea temporală și spațială între IoT și nivelul Cloud** – cu scopul de a proteja funcționalitățile de timp real ale nodurilor de la nivelul Edge, față de comportamentul care nu este de timp real al nivelului Cloud, sau al cererilor transmise de către utilizatori.
- **Monitorizarea aplicațiilor** – care rulează la nivel de Edge.
- **Agregarea datelor** – agregarea datelor recepționate de la nodurile IoT, procesarea acestor date local, apoi transmiterea lor către Cloud, reducând astfel comunicarea între nivelurile Edge și Cloud.

3.2.1.3 Exemple de arhitecturi bazate pe Fog în IoT

Există mai multe domenii în care dispozitivele IoT pot beneficia de arhitecturile bazate pe Fog [160]:

A. Vehicule autonome

VIoT (Vehicular Internet of Things) [161] presupune comunicarea și schimbul de date între vehicule, infrastructuri de trafic sau alte entități ce țin de sistemele inteligente de transport. În următorii ani se așteaptă o creștere a numărului de vehicule conectate la rețea și care sunt capabile să comunice cu alte vehicule. Pentru a asigura interacțiunea în timp real, cea mai eficientă soluție într-o astfel de infrastructură este o arhitectură bazată pe Fog.

Conform [162, 163] monitorizarea condițiilor din trafic se va realiza prin intermediul a două tipuri de noduri Fog: staționare sau mobile (transportate de către vehicule purtătoare). Acestea prezintă capabilități de procesare și stocare a informațiilor transmise de către alte vehicule din rețea. Astfel, nodurile Fog pot oferi servicii de navigare în timp real prin cooperarea cu celelalte noduri din rețea, reducând astfel numărul de coliziuni și accidente din trafic [164].

B. Sisteme de semaforizare inteligente

Sistemele de transport inteligente au fost dezvoltate pentru a controla în mod dinamic fluxul de trafic în funcție de condițiile din timp real. Într-o astfel de infrastructură, sistemul clasic de semaforizare, bazat pe intervale fixe de timp, devine ineficient. Introducerea conceptului de Fog în controlul sistemele de semaforizare inteligente va permite adaptarea dinamică a timpilor (perioadele de verde, roșu și galben) în funcție de poziția, viteza și direcția vehiculelor [165, 166].

C. Case inteligente

O casă inteligentă incorporează numeroase dispozitive și senzori IoT, precum [167]: controlul și automatizarea sistemelor de iluminat, ventilație, aer condiționat, purificatoare de aer, electrocasnice conectate la rețeaua fără fir pentru monitorizarea la distanță, etc. Aceste dispozitive provin de la diverși producători, în consecință prezintă platforme diferite.

Arhitecturile bazate pe Fog furnizează o interfață unificată în care pot fi integrate diferite dispozitive și echipamente IoT. Aceste infrastructuri prezintă avantaje majore, mai ales în cazul aplicațiilor pentru securitatea locuinței, întrucât oferă resurse flexibile pentru stocare, procesare în timp real și latență scăzută [168, 169].

D. Dispozitive medicale și de îngrijire medicală

În aplicațiile medicale și de îngrijire medicală critice este esențial ca procesarea datelor să se realizeze în timp real. Interacțiunea unui număr mare de dispozitive și echipamente medicale pentru stocarea la distanță, procesarea și preluarea datelor medicale din Cloud necesită o conexiune stabilă la rețea, care în prezent nu este disponibilă. Arhitecturile bazate pe Fog pot rezolva problemele ce țin de conectivitatea la rețea și traficul de date [170, 171]. Nodurile Fog oferă metode de monitorizare de la distanță a sănătății, stocare distribuită, precum și servicii de notificare de urgență [172].

E. Rețele fără fir de senzori și actuatoare

Unul din avantajele rețelelor fără fir de senzori este capacitatea de a rula la putere mică ceea ce crește durata de viață a bateriilor. Actuatoarele sunt folosite pentru administrarea operațiile de măsurare și generarea de acțiuni, în urma informațiilor recepționate de la senzori. Acestea pot fi văzute ca noduri Fog ce controlează dispozitivele IoT cu ajutorul senzorilor [173, 174]. Rețelele fără fir de senzori și actuatoare necesită mai puțină lățime de bandă, energie și o putere de procesare mică [175].

F. Sisteme cyber physical și IoT

IoT este o rețea capabilă să interconecteze dispozitive diferite prin intermediul Internetului și a telecomunicațiilor, în timp ce CPSs (Cyber Physical Systems) combină componentele fizice (mediul înconjurător) cu sistemele de calcul. Întrucât arhitecturile bazate pe Fog sunt dezvoltate luând în considerare noțiunea de sisteme incorporate (vehicule autonome, dispozitive medicale, etc.), prin integrarea acestora în IoT și CPSs [176], va deveni posibilă dezvoltarea de dispozitive medicale inteligente [139], clădiri inteligente [177], sisteme robotice distribuite, etc.

G. Realitatea augmentată

Realitatea augmentată (AR) se referă la sisteme interactive bazate pe combinarea lumii reale cu elemente virtuale, rezultând astfel o lume reală augmentată de către informația generată virtual [178]. Aplicațiile AR prezintă un nivel ridicat de sensibilitate la latență, întrucât orice întârziere în răspunsul aplicației va deteriora experiența utilizatorului. Nodurile Fog pot juca un rol important într-un astfel de sistem deoarece asigură procesarea datelor în timp real.

Un exemplu de aplicație AR bazată pe arhitectura Fog este cea introdusă în [179], sub forma unui joc de tipul „Augmented Brain-Computer Interaction" (ABCI). Astfel, prin combinarea servelor Fog și Cloud se poate obține o interacțiune cu utilizatorul în timp real, continuă.

H. Rețele inteligente (Smart Grid)

Rețelele smart grid oferă o distribuție a energiei care poate fi monitorizată și controlată atât de către furnizori, cât și de către clienți cu ajutorul contoarelor inteligente [180]. Într-o astfel de infrastructură datele sunt stocate la nivel de Cloud. Din această cauză rețelele smart grid trebuie să asigure confidențialitate, fiabilitate, securitate, flexibilitate și scalabilitate. Arhitecturile bazate pe Fog reprezintă o soluție eficientă pentru rezolvarea unor probleme precum: procesarea datelor utilizatorilor mobili [181], furnizarea de servicii „delay-sensitive" și de percepere a locației [182], îmbunătățirea performanței site-urilor web [183] și administrarea eficientă a informațiilor din rețelele IoT [184]. Păstrarea datelor private la nivel local permite accesarea lor într-un mod rapid și sigur de către clienți.

3.2.2 Arhitectura Fog MC-IoT propusă

Sistemele cu niveluri mixte de criticalitate reprezintă o clasă specială de sisteme de timp real, în care aplicații de criticalitate diferită împart aceleași resurse computaționale, și/sau de comunicare, hardware (magistrale, senzori, memorie), software (mutex, date de prelucrat) [17]. Conceptul platformelor cu niveluri mixte de criticalitate oferă avantaje importante atunci când este integrat în diferite tipuri de sisteme care interacționează direct cu mediul înconjurător [17]. Din această categorie fac parte și sistemele IoT, acestea interacționând direct cu utilizatorul/mediul

înconjurător. Sistemele IoT prezintă o evoluție rapidă înspre sisteme IoT de timp real, în care task-uri de criticalitate diferită și cu diverse constrângeri temporale coexistă. Deoarece planificarea în astfel de platforme devine din ce în ce mai complexă, integrarea conceptul de sisteme cu niveluri mixte de criticalitate devine o necesitate.

Adaptarea conceptului de sisteme cu niveluri mixte de criticalitate în arhitecturile IoT, dă naștere unei noi paradigme, denumită MC-IoT, pentru care teza de față propune următoarea definiție:

Definiția 1. *Mixed Criticality-Internet of Things (MC-IoT) sunt sisteme care rulează task-uri de timp real de criticalitate diferită în cadrul nivelului Edge al arhitecturilor IoT.*

Infrastructura arhitecturilor bazate pe Cloud [185] este implementată în centre de date, în mare parte, din motive economice. Aici datele provenite de la dispozitivele IoT sunt stocate în Cloud, unde vor fi procesate și se vor lua decizii referitoare la acțiunile care trebuie efectuate. Dacă numărul nodurilor Edge crește, sau dacă se afla la o distanță mare în rețea față de centrele de date, procesul de luare a deciziilor va prezenta o latență ridicată. În consecință, arhitecturile bazate pe Cloud nu pot fi utilizate pentru aplicații „time-sensitive” (de timp real).

Întrucât această teză abordează domeniul sistemelor distribuite de timp real, arhitecturile bazate pe Fog [144] sunt cele mai potrivite deoarece procesarea datelor are loc în apropierea dispozitivelor IoT. Scopul acestor arhitecturi este de a oferi servicii de procesare, stocare și comunicație pentru nodurile Edge astfel încât nivelul Cloud să fie utilizat într-un mod cât mai eficient.

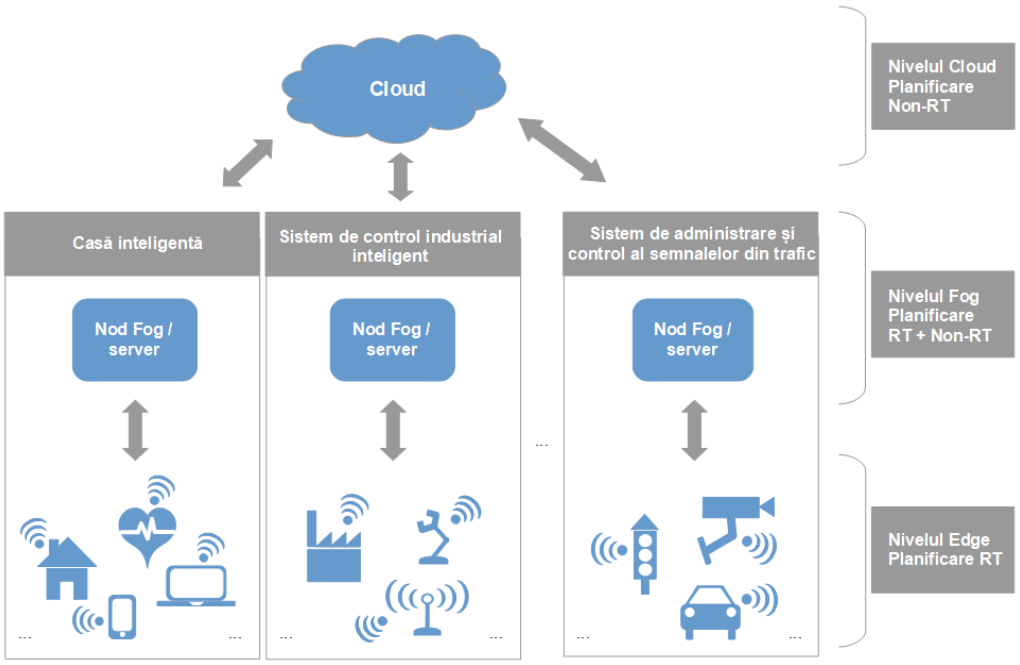


Figura 6. Arhitectura Fog MC-IoT propusă.

În arhitecturile bazate pe Fog, precum cea descrisă în Figura 6, nivelul Edge poate fi văzut drept o colecție de rețele distribuite eterogene multiple, conectate la Cloud cu ajutorul unui nivel intermediar denumit Fog. În arhitectura propusă, nivelul Fog este puternic interconectat cu nivelul Edge, dar, de asemenea, acționează ca un intermediar între nivelurile Cloud și Edge.

Astfel, sunt considerate una sau mai multe rețele Fog centralizate eterogene conectate la nivelul Cloud: casă inteligentă, sistem de control industrial, sistem de administrare și control al semnalelor din trafic, etc. Fiecare rețea are un nod Fog care se ocupă cu procesarea și stocarea datelor primite de la nodurile Edge din rețeaua respectivă. Aceste noduri pot comunica între ele, dar și cu dispozitivele Fog. Nodurile Edge sunt reprezentate de către orice tip de dispozitiv capabil de procesare [13]:

- Casă inteligentă – dispozitive multimedia, de securitate, de monitorizare a sănătății, etc.
- Sistem de control industrial – dispozitive robotice, uzine, senzori inteligenți, etc.
- Sistem de administrare și control al semnalelor din trafic – semafoare, automobile, dispozitive de monitorizare a traficului, etc.

Metodele utilizate pentru planificarea task-urilor variază în funcție de nivelul arhitecturii. Astfel, spre deosebire de Cloud, care nu oferă suport pentru aplicațiile „time-sensitive”, nivelul Fog prezintă funcționalități care să permită planificarea în timp real a anumitor task-uri.

Nodurile din rețeaua Edge pot rula aplicații de timp real împărțite în task-uri de criticalitate diferită, rezultând astfel o rețea Edge cu niveluri mixte de criticalitate. Task-urile vor fi planificate la nivel de element de procesare (PE: Processing Element). În această arhitectură fiecare element de procesare este reprezentat de un sistem cu o singură unitate de procesare.

Pe baza arhitecturii propuse, descrisă în Figura 6, în continuare este prezentat un model de execuție al task-urilor și o formalizare matematică a problemei de planificare.

3.2.2.1 Formalizarea problemei

În cadrul arhitecturii propuse aplicațiile sunt planificate pe diferite niveluri. O parte din servicii și cereri sunt procesate la nivelul Cloud, o parte la nivelul Fog și o parte din aplicațiile împărțite în task-uri sunt planificate atât la nivelul Fog, cât și la nivelul Edge.

Aplicațiile care rulează la nivelul Edge sunt împărțite în unități de execuție de bază, denumite task-uri. Task-urile pot fi independente unul față de celălalt sau pot exista anumite dependențe între acestea (de prioritate, resurse, evenimente, etc.). Fiecare task, sau set de task-uri dependente va fi alocat unui nod specific printr-un proces denumit planificare globală. Apoi, fiecare set de task-uri independente sau dependente este planificat la nivel de dispozitiv, printr-un proces care poartă numele de planificare locală.

Prin urmare, problema planificării la nivelul Edge poate fi împărțită în două subprobleme (Figura 7):

1) Maparea task-urilor (la nivelul Fog) – fiecare task trebuie să fie alocat/mapat pe un element de procesare.
2) Planificarea locală a task-urilor (la nivel de dispozitiv Edge) – toate task-urile alocate pe un element de procesare trebuie să fie planificabile.

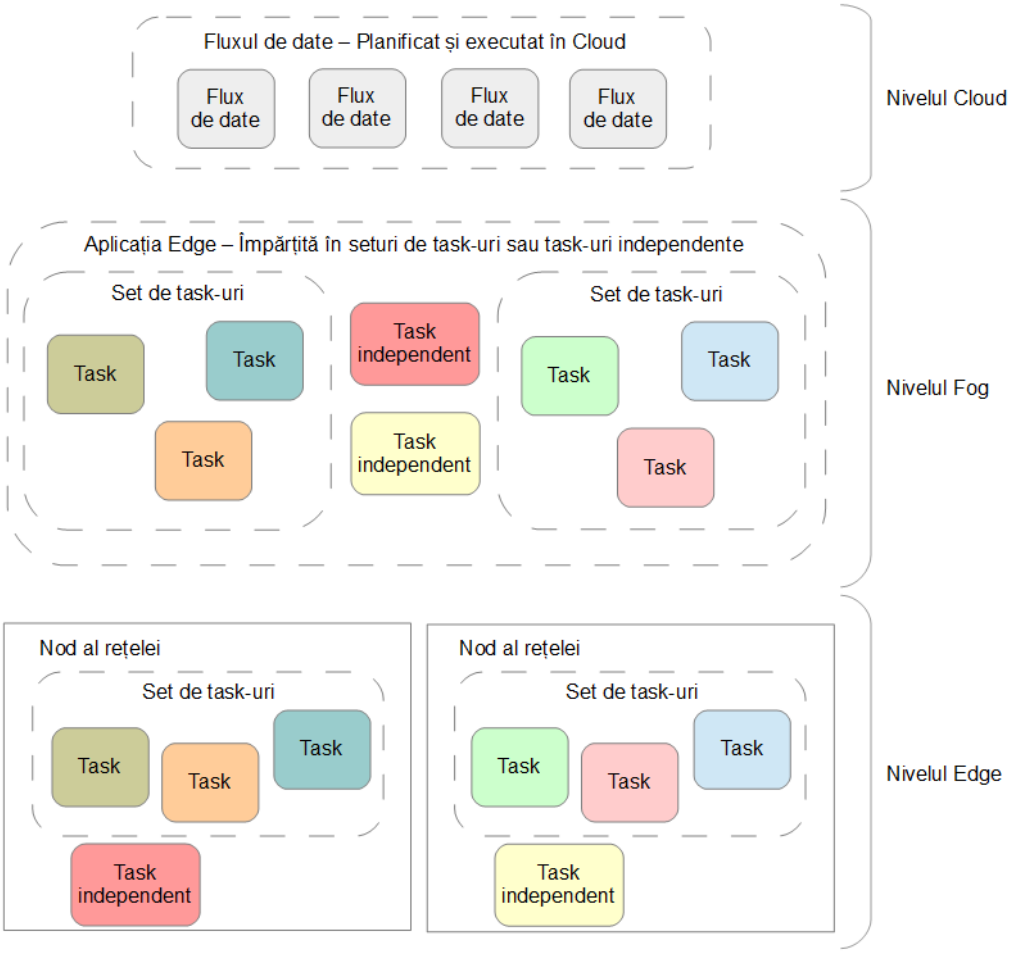


Figura 7. Planificarea aplicațiilor în cadrul arhitecturii Fog MC-IoT propuse.

3.2.2.2 Ipoteze de lucru

În continuare sunt folosite următoarele ipoteze de lucru:

- Aplicațiile sunt implementate sub forma unui sistem de task-uri periodice/sporadice de criticalitate diferită.
- Task-urile prezintă constrângeri temporale stricte (cu alte cuvinte, un astfel de sistem nu tolerează nici un DM (Deadline Miss)).
- Timpul de execuție pentru cazul cel mai defavorabil (WCET: Worst Case Execution Time) este determinat prin analiză statică pentru nivelurile de criticalitate ridicate și estimat prin măsurători pentru nivelurile de criticalitate scăzute.
- Fiecare task este alocat static unui element de procesare și nu poate să migreze pe un alt element de procesare în timpul rulării sistemului.
- Modul de rulare al fiecărui element de procesare este setat la început pe nivelul de criticalitate cel mai scăzut L_1, ceea ce înseamnă că doar task-

urile care au nivelul de criticalitate $L_j \geq L_1$ vor fi executate. Astfel, elementul de procesare va rămâne în modul L_1 atât timp cât toate instanțele task-urilor sunt executate în limita timpului de execuție al nivelului de criticalitate respectiv, C_{i,L_1}. Dacă timpul de execuție al unei instanțe depășește C_{i,L_1} înainte de a semnala finalizare, sistemul va trece în modul L_2, și așa mai departe [30].

- Pentru a simplifica lucrurile, în cazul fiecărui mod de rulare L_j, se va renunța la task-urile de criticalitate mai scăzută decât L_j [30]. Atât modelele, cât și algoritmii de partiționare propuși în această teză sunt aplicabili și în cazul algoritmilor mai complicați de planificare locală a task-urilor, în care nu se renunță la task-urile de criticalitate mai scăzută. Aceste task-uri sunt, în schimb, planificate folosind așa numitul „slack time" al procesorului.
- Sistemul este eterogen, ceea ce înseamnă că timpul de execuție al task-urilor variază în funcție de fiecare element de procesare.

4 O METODOLOGIE DE MAPARE A TASK-URILOR PE DIFERITE ELEMENTE DE PROCESARE

4.1 Modelul de task-uri propus pentru sisteme distribuite cu niveluri mixte de criticalitate

Pornind de la modelul clasic pentru sisteme cu niveluri mixte de criticalitate introdus de Vestal în 2007 [30] și luând în considerare comportamentul temporal exprimat prin intermediul expresiei (2-1), această teză introduce un nou parametru: scorul de afinitate. A_i caracterizează afinitatea task-ului i pentru fiecare element de procesare din sistem și este definit ca un vector de valori, câte o valoare pentru fiecare element de procesare. Scorul de afinitate este un întreg cuprins între 0 și p, unde p reprezintă numărul de elemente de procesare (PE). O valoare mai mare înseamnă o afinitate mai mare, în timp ce 0 semnifică lipsa afinității.

Astfel, timpul de execuție va fi exprimat atât în funcție de nivelul de criticalitate (L_i), cât și în funcție de elementul hardware de procesare pe care rulează task-ul (PE_q). Prin urmare, modelul de task-uri (2-1) devine:

$$\tau_i = \left\{T_i, D_i, L_i, \left\{C_{i,L_j\,PE_q} \mid j \in 1 \dots l, q \in 1 \dots p\right\}, \left\{A_{i,PE_q} \mid q \in 1 \dots p\right\}\right\} \qquad (4\text{-}1)$$

unde:

- C_i este o matrice de dimensiune $p \times l$ (p reprezintă numărul de elemente de procesare și l, numărul de niveluri de criticalitate).

Scorul de afinitate poate fi setat în mod static de către cel care a creat task-urile, sau calculat folosind un algoritm. Acest algoritm trebuie să se bazeze pe resursele necesare și pe timpul de execuție al task-ului în cazul fiecărui element de procesare. Dacă un task nu poate fi planificat pe un anumit element de procesare, valoarea WCET-ului poate fi setată ca fiind infinită, ceea ce se traduce printr-un scor de afinitate egal cu 0.

În această teză se consideră un sistem cu două niveluri de criticalitate (Lo – criticalitate scăzută și Hi – criticalitate ridicată), dar algoritmul poate fi aplicat și pe platforme cu mai multe niveluri de criticalitate. În cadrul euristicilor de partiționare a task-urilor pentru sisteme cu niveluri mixte de criticalitate este de preferat să se folosească utilizarea în modul Hi, în timpul alocării task-urilor de criticalitate Hi pe un anumit element de procesare q (U_{τ_i,L_{Hi},PE_q}) și utilizarea în modul Lo, în timpul alocării task-urilor de criticalitate Lo (U_{τ_i,L_{Lo},PE_q}). Conform acestui principiu, scorul de afinitate pentru task-urile de criticalitate Hi este setat folosind WCET-ul pentru nivelul de criticalitate Hi, iar pentru task-urile de criticalitate Lo, folosind WCET-ul pentru nivelul de criticalitate Lo.

Folosind modelul de task-uri definit anterior, precum și problema planificării formalizată în Secțiunea 3.2.2, această teză propune o metodologie de mapare a task-urilor pentru sisteme distribuite cu niveluri mixte de criticalitate. Metodologia cuprinde diferite metode de setare a noului parametru introdus, adică a scorului de afinitate și de definire a unei funcții de mapare adecvată, astfel încât să fie respectate cerințele aplicației și constrângerile ce țin de resurse.

4.2 Setarea scorului de afinitate

În timp ce parametrii task-urilor, moșteniți de la modelul clasic pentru sisteme cu niveluri mixte de criticalitate, iau în considerare doar comportamentul temporal [30], scorul de afinitate este calculat în funcție de particularitățile fiecărui element de procesare.

4.2.1 Setarea scorului de afinitate pe baza timpului de execuție

O metodă pentru setarea scorului de afinitate pe baza timpului de execuție este prezentată sub formă de pseudocod în Algoritm 1:

Algoritm 1. SetAffinity_WCET.

Input: $C_{i,L_2 PE_q}$
Output: A_{i,PE_q}
1 **for** $i \in \{1, 2, \dots, n\}$ **do**
2 **for** $q \in \{1, 2, \dots, p\}$ **do**
3 $X_{i,L_2 PE_q} \leftarrow C_{i,L_2 PE_q}$
4 **end for**
5 **end for**
6 **for** $i \in \{1, 2, \dots, n\}$ **do**
7 $a \leftarrow 1$
8 **for** $q \in \{1, 2, \dots, p\}$ **do**
9 $index \leftarrow \max(X_{i,L_2 PE_q})$
10 $A_{i,PE_{index}} \leftarrow a$
11 $a \leftarrow a + 1$
12 $X_{i,L_2 PE_{index}} \leftarrow 0$
13 **end for**
14 **end for**

Etapa 1: Se extrage timpul de execuție al task-urilor pentru nivelul de criticalitate cel mai ridicat pe fiecare element de procesare (a doua coloană din matricea generată ce conține timpii de execuție $C_{i,L_j PE_q}$), prin copierea acestuia într-un tablou de structuri $X_{i,L_2 PE_q}$. Fiecare structură conține o valoare pentru timpul de execuție $X_{i,L_2 PE_q}$ și un index ce corespunde elementului de procesare (q), unde i reprezintă indexul task-ului și este fix, iar L_2 este nivelul de criticalitate cel mai ridicat. q variază de la 1 la p.

Etapa 2: Se extrage indexul liniei matricei ce conține valoarea maximă pentru $X_{i,L_2 PE_q}$ din tabloul de structuri, în timp ce q variază de la 1 la p.

Etapa 3: În tabloul ce conține afinitatea față de fiecare element de procesare, scorul de afinitate A_{i,PE_q} se setează pe 1 pentru elementul de procesare ce corespunde valorii maxime $X_{i,L_2 PE_q}$, iar apoi pe 2 pentru elementul de procesare ce corespunde celei mai mari valori $X_{i,L_2 PE_q}$ din tablou după setarea elementului cu valoarea maximă

din prima iterație $X_{i,L_2\,PE_{index}}$ pe 0, 3 pentru următoarea valoare maximă și așa mai departe, în timp ce $q \leq p$.

4.2.2 Setarea scorului de afinitate pe baza nivelului de criticalitate

Pentru a lua în considerare nivelul de criticalitate la distribuirea task-urilor pe diferite elemente de procesare, se pot aplica metodele descrise în [186].

O variantă alternativă pentru maparea task-urilor, care ia în considerare scorul de afinitate, este descrisă în continuare. Dacă numărul de elemente de procesare este mai mare decât numărul de niveluri de criticalitate:

Algoritm 2. SetAffinity_criticality (p>l).

Input: L_i, C_{i,L_2PE_q}
Output: A_{i,PE_q}

for $i \in \{1, 2, \dots, n\}$ **do**
  for $q \in \{1, 2, \dots, p\}$ **do**
    $X_{i,L_2PE_q} \leftarrow C_{i,L_2PE_q}$
  end for
end for
for $q \in \{1, 2, \dots, p\}$ **do**
  $PE_{q,L_j} \leftarrow q \; mod \; l$
  if $q \; mod \; l = 0$ **then**
    $PE_{q,L_j} \leftarrow l$
  end if
end for
for $i \in \{1, 2, \dots, n\}$ **do**
  $a \leftarrow 0$
  for $q \in \{1, 2, \dots, p\}$ **do**
    $\{maxVal, index\} \leftarrow \max\,(X_{i,L_2PE_q} | PE_{q,L_j} \sim = L_i)$
    if $maxVal \sim = 0$ **then**
      $a \leftarrow a + 1$
      $A_{i,PE_{index}} \leftarrow a$
      $X_{i,L_2PE_{index}} \leftarrow 0$
    end if
  end for
end for
for $i \in \{1, 2, \dots, n\}$ **do**
  $a2 \leftarrow p - a$
  for $q \in \{1, 2, \dots, p\}$ **do**
    $\{maxVal, index\} \leftarrow \max\,(X_{i,L_2PE_q} | PE_{q,L_j} = L_i)$
    if $maxVal \sim = 0$ **then**
      $a2 \leftarrow a2 + 1$
      $A_{i,PE_{index}} \leftarrow a2$
      $X_{i,L_2PE_{index}} \leftarrow 0$

```
31          end if
32      end for
33  end for
```

Etapa 1: Se construiește un tablou de structuri PE_{q,L_j}. Fiecărei structuri îi corespunde un nivel de criticalitate L_j, reprezentând nivelul de criticalitate al task-urilor care vor fi partiționate pe PE_q, precum și un index q al elementului de procesare. Indexul variază de la 1 la p. L_j este calculat cu formula $L_j = PE_q \ mod \ l$, unde l reprezintă numărul de niveluri de criticalitate, iar mod este operația modulo. Dacă L_j are valoarea 0, atunci se va considera $L_j = l$.

Etapa 2: Se alocă fiecare task τ_i în funcție de nivelul de criticalitate L_i. Astfel, rezultă două subseturi: elementele de procesare PE_{q,L_j} care așteaptă task-uri de criticalitate egală cu L_i și elementele de procesare PE_{q,L_j} care așteaptă task-uri de criticalitate diferită față de L_i. Scorul de afinitate A_{i,PE_q} va avea cele mai mari valori în cazul primului subset de elemente de procesare. Pentru fiecare subset de elemente de procesare, afinitatea este stabilită în funcție de timpul de execuție, asemănător primului algoritm (Algoritm 1).

Dacă numărul de niveluri de criticalitate este mai mare decât numărul de elemente de procesare:

Algoritm 3. SetAffinity_criticality (l>p).

Input: L_i, C_{i,L_2PE_q}
Output: A_{i,PE_q}

```
1   for i ∈ {1, 2, ..., n} do
2       for q ∈ {1, 2, ..., p} do
3           X_{i,L_2PE_q} ← C_{i,L_2PE_q}
4       end for
5   end for
6   for i ∈ {1, 2, ..., n} do
7       q ← L_i mod p
8       if q = 0 then
9           q ← p
10      end if
11      A_{i,PE_q} ← p
12      X_{i,L_lPE_q} ← 0
13  end for
14  for i ∈ {1, 2, ..., n} do
15      a ← 0
16      for q ∈ {1, 2, ..., p − 1} do
17          index ← max (X_{i,L_lPE_q})
18          a ← a + 1
19          A_{i,PE_index} ← a
20          X_{i,L_lPE_index} ← 0
21      end for
```

22 **end for**

Etapa 1: Pentru fiecare task τ_i, se obține elementul de procesare pe care trebuie să ruleze folosind formula $PE_q = L_i \, mod \, p$, unde p este numărul de elemente de procesare și mod reprezintă operația modulo. Dacă PE_q este egal cu 0, se consideră $PE_q = p$. Apoi, se setează scorul de afinitate A_{i,PE_q} pentru PE_q pe p.

Etapa 2: Pentru fiecare task τ_i, se stabilesc scorurile de afinitate rămase A_{i,PE_q} conform timpului de execuție (Algoritm 1), unde q variază de la 1 la $p - 1$.

4.3 Maparea task-urilor

Se dă un set de task-uri independente de criticalitate diferită și un set de elemente de procesare. Trebuie să se găsească o funcție astfel încât:

$$M: \tau \rightarrow P \tag{4-2}$$

unde:

- τ este setul de task-uri din sistem;
- P reprezintă setul de elemente de procesare (PEs);
- M exprimă un morfism.

Luând în considerare ipoteza de lucru prezentată anterior, task-urile nu pot migra în timpul rulării, deci fiecare task este alocat unui singur element de procesare. $M(\tau_i)$ este elementul de procesare pe care va rula task-ul τ_i. Figura 8 ilustrează modul în care task-urile sunt partiționate pe fiecare element de procesare: p indică numărul total de elemente de procesare, iar n exprimă numărul de task-uri din sistem.

După terminarea partiționării, task-urile alocate unui element de procesare PE_q formează un subset de task-uri, definit Ψ_q [54]. Dacă setul de task-uri τ este partiționat cu succes atunci:

$$\tau = \Psi_1 \cup \Psi_2 \cup \Psi_3 \cup \ldots \cup \Psi_p \tag{4-3}$$

unde:

- Ψ_q poate fi și un subset gol $\{\emptyset\}$.

Ideea este de a găsi o funcție M potrivită, astfel încât să se respecte următoarele cerințe, ordonate în funcție de prioritate:

I. Crearea unor subseturi de task-uri Ψ_q, astfel încât fiecare subset să fie planificabil pe elementul de procesare pe care a fost alocat (PE_q), de către algoritmul de planificare local.
II. Respectarea scorului de afinitate al task-ului.
III. Optimizarea utilizării resurselor.

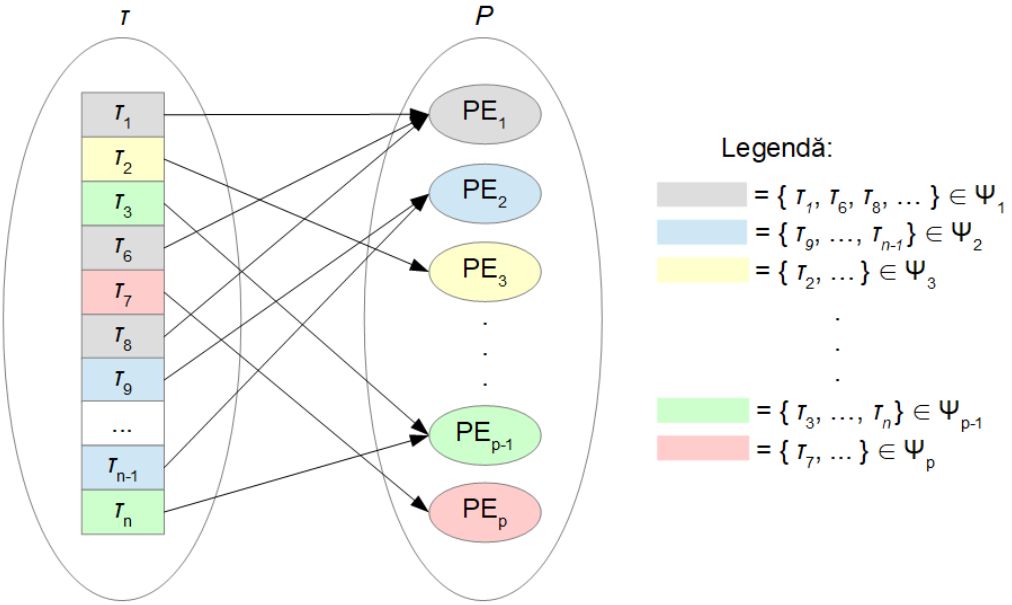


Figura 8. Alocarea task-urilor la elementele de procesare.

4.4 Planificarea locală a task-urilor

Conform lucrării lui Vestal [30], un set de task-uri este planificabil dacă deadline-ul fiecărui task τ_i este mai mare sau egal cu timpul de răspuns pentru cazul cel mai defavorabil:

$$R_i \leq D_i \tag{4-4}$$

unde:

- R_i reprezintă durata maximă între momentul de terminare și momentul de activare pentru fiecare job al task-ului τ_i [68]:

$$R_i = C_i + I_i \tag{4-5}$$

unde:

- I_i indică influența indusă de la task-urile de prioritate mai ridicată.

Valoarea lui R_i depinde de algoritmul de planificare și poate fi obținută în mod iterativ, calculând cel mai mic punct fix al relației (4-6) [30]:

$$R_i = \sum_{\tau_j \epsilon hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{4-6}$$

unde:

- $hp(i)$ este un subset de task-uri de prioritate mai mare sau egală cu cea a task-ului τ_i;
- C_i exprimă timpul de execuție pentru cazul cel mai defavorabil;
- T_i indică perioada task-ului.

Planificarea în sisteme cu niveluri mixte de criticalitate este fezabilă doar dacă următoarele două condiții sunt îndeplinite [187]:

Condiția 1: Dacă timpul de execuție pentru toate job-urile nu depășește valoarea WCET-ului C_{i,L_j} în modul de criticalitate curent L_j, atunci toate job-urile care au nivelul de criticalitate L_i mai mare sau egal decât L_j trebuie să fie finalizate înainte de deadline.

Condiția 2: Dacă cel puțin un job depășește valoarea timpului de execuție C_{i,L_j} în modul de criticalitate curent L_j, atunci toate job-urile care au nivelul de criticalitate mai mare decât L_j trebuie să fie finalizate înainte de deadline, în schimb se poate renunța la job-urile care au nivelul de criticalitate mai mic sau egal decât L_j.

Mai mult decât atât, pentru ca toate deadline-urile să fie îndeplinite, elementele de procesare nu trebuie să fie supraîncărcate cu task-uri. În consecință, o condiție **necesară**, dar nu și suficientă, legată de metrica de încărcare pentru fiecare element de procesare este dată de [187]:

Pentru fiecare $q = 1 \dots p$:

$$U_{L_j \Psi_q} \leq 1, \;\; j = 1 \dots l \tag{4-7}$$

În cazul task-urilor periodice/sporadice utilizarea totală pe fiecare element de procesare în modul de rulare L_j este:

$$U_{L_j \Psi_q} = \sum_{\tau_i \epsilon hc(L_j), k=j}^{k \leq l} \frac{C_{iL_k PE_q}}{T_i} \tag{4-8}$$

unde:

- $hc(L_j)$ este un subset de task-uri din Ψ_q, fiecare task având nivelul de criticalitate mai mare sau egal decât L_j;
- l reprezintă numărul de niveluri de criticalitate;
- $L_{k=j \dots l}$ sunt nivelurile de criticalitate mai mari sau egale decât L_j;
- $C_{iL_k PE_q}$ indică WCET-ul task-ului i, care rulează în modul de criticalitate k, pe elementul de procesare PE_q.

Mulți algoritmi de planificare locali au condiții de suficiență. De exemplu, condiția de **suficiență** pentru algoritmul de planificare local denumit EDF-VD, în cazul unul sistem cu două niveluri de criticalitate este dată de [54]:

$$\max\left(U_{L_1 \Psi_q}, U_{L_2 \Psi_q}\right) \leq \frac{3}{4} \tag{4-9}$$

4.5 Deviația totală a scorului de afinitate

Scorul de afinitate al task-urilor poate fi setat în mod static de către cel care a creat setul de task-uri, sau calculat cu ajutorul unui algoritm. Algoritmul trebuie să ia în considerare resursele necesare fiecărui task, precum și timpul de execuție al task-urilor pentru fiecare element de procesare. Atât timp cât condițiile de planificare locale nu sunt încălcate, maparea se va realiza în funcție de scorul de afinitate al fiecărui task. Astfel, această teză propune o formulă prin care se poate calcula deviația totală a scorului de afinitate $A_{d\tau}$ pentru o anumită mapare:

$$A_{d\tau} = \sum_{i=1}^{n} \left(p - A_{i,PE_q} \right) \tag{4-10}$$

unde:

- q este indexul subsetului Ψ, ce conține task-ul i;
- p indică numărul de elemente de procesare (p este, de asemenea, egal cu cea mai mare valoare a scorului de afinitate).

Scopul este de a minimiza valoarea lui $A_{d\tau}$, respectând condiția de planificare impusă de către algoritmul de planificare local (condiția I din Subcapitolul 4.3).

4.6 Optimizarea utilizării resurselor

Atunci când vorbim despre optimizarea utilizării resurselor se ia în considerare o varietate largă de resurse, precum: numărul de procesoare, utilizarea totală a energiei electrice, utilizarea totală a memoriei, etc. Un exemplu tipic de optimizare a resurselor este utilizarea unui număr cât mai mic de elemente de procesare, astfel încât elementele de procesare care nu sunt folosite să poată trece pe modul de economisire a energiei electrice (idle). În continuare este prezentat un exemplu care abordează optimizarea numărului de elemente de procesare utilizate.

Astfel se introduce notația Ψ', având următoarea semnificație: Ψ' conține subseturile de task-uri alocate fiecărui element de procesare care nu sunt egale cu mulțimea vidă.

$$\Psi' = \left\{ \bigcup_{q=1}^{q \leq p} \Psi_q \,|\Psi_q \neq \{\emptyset\} \right\} \tag{4-11}$$

Scopul este de a minimiza numărul de subseturi de task-uri din Ψ', respectând condițiile de planificare impuse de către algoritmul de planificare local și a scorului de afinitate (condițiile I și II din Subcapitolul 4.3).

4.7 Funcția de mapare propusă – Best Affinity Fit

Această teză introduce un nou algoritm de mapare, denumit Best Affinity Fit (BAF), care ia în considerare valoarea afinității la partiționarea task-urilor pe

elementele de procesare din sistem. Algoritmul se bazează pe metodele de setare a scorului de afinitate introduse la începutul acestui capitol.

Algoritm 4. Best Affinity Fit (BAF).

Input: τ_i
Output: Ψ_q, $U_{L_j\Psi_q}$

for $i \in \{1, 2, \dots, n\}$ **do**
  $assign \leftarrow 1$
  $\{maxVal, index\} \leftarrow \max(A_{i,PE_q})$
  while $U_{L_j\Psi_{index}} + C_{i,L_jPE_{index}}/T_i > 1$ **do**
    $A_{i,PE_{index}} \leftarrow 0$
    $\{maxVal, index\} \leftarrow \max(A_{i,PE_q})$
    if $maxVal = 0$ **then**
      $assign \leftarrow 0$
    end if
  end while
  if $assign \leftarrow 1$ **then**
    add τ_i to Ψ_{index}
    $U_{L_j\Psi_{index}} \leftarrow U_{L_j\Psi_{index}} + C_{i,L_jPE_{index}}/T_i$
  end if
end for

Pentru fiecare task τ_i:

Etapa 1: Se presupune că task-ul poate fi alocat pe un element de procesare, din această cauză variabila $assign$ este inițializată cu valoarea 1. În continuare se găsește elementul de procesare cu cel mai mare scor de afinitate pentru task-ul τ_i. De asemenea, utilizarea elementului de procesare trebuie să respecte formula (4-7)).

Etapa 2: Dacă un element de procesare PE_{index} nu poate să accepte task-ul i, scorul de afinitate $A_{i,PE_{index}}$ este setat pe 0. În plus, dacă nu există nici un element de procesare pe care să poată fi alocat task-ul τ_i, $assign$ va primi valoarea 0.

Etapa 3: Dacă, în schimb, s-a găsit un element de procesare PE_{index} pe care să poată fi alocat, se adaugă task-ul τ_i în subsetul Ψ_{index}, iar utilizarea elementului de procesare este actualizată.

5 MEDIUL DE SIMULARE ȘI EVALUARE A PERFORMANȚELOR

5.1 Structura mediului de simulare și evaluare

Algoritmul de mapare a task-urilor Best Affinity Fit (BAF) a fost, inițial, implementat în MATLAB, pentru testarea sa și compararea cu cei doi algoritmi: Best Fit Decreasing Utilization (BFDU) și Best Fit Decreasing Criticality (BFDC).

După efectuarea simulărilor în MATLAB, atât modelul de task-uri, cât și algoritmul de mapare au fost integrate cu succes într-un mediu de simulare dezvoltat în C++ din literatura de specialitate pentru sisteme omogene cu niveluri mixte de criticalitate [188]. Simulatorul astfel modificat poate fi utilizat pentru sisteme eterogene distribuite cu niveluri mixte de criticalitate. Modul de funcționare al mediului de simulare este descris în Figura 9 și explicat în următoarele secțiuni.

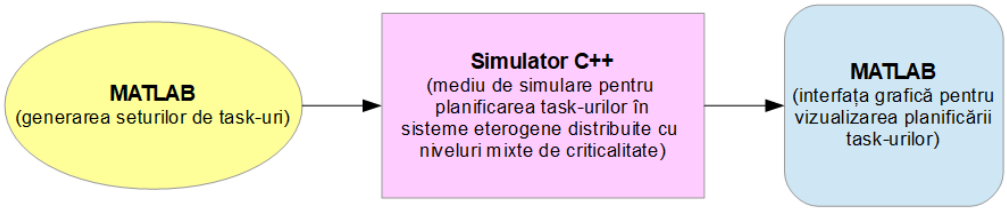


Figura 9. Modul de funcționare al mediului de simulare.

5.2 Generarea seturilor de task-uri

Generarea seturilor de task-uri s-a realizat în MATLAB. Toate task-urile au fost generate aleatoriu, utilizând algoritmul introdus în [189] și care are la bază metoda lui Guan [190]. Ca parametrii de intrare pentru algoritm se vor introduce:

- $[a, b]$: Domeniul de valori în care este generată uniform perioada.
- P_{Hi}: Probabilitatea ca un task să fie de criticalitate Hi, unde $0 \leq P_{Hi} \leq 1$.
- Pentru generarea utilizărilor (gradele de utilizare ale fiecărui element de procesare), se iau în considerare cinci parametri de intrare [189]:
  - U_{bound}: Limita superioară a utilizării totale pentru setul de task-uri, obținută folosind relația (5-1).

$$max\left(U_{Lo}(\tau), U_{Hi}(\tau)\right) = U_{bound} \tag{5-1}$$

$$U_{Lo}(\tau) = \sum_{\tau_i \in \pi} U_{i,L_{Lo}} \tag{5-2}$$

$$U_{Hi}(\tau) = \sum_{\tau_i \in Hi(\pi)} U_{i,L_{Hi}} \tag{5-3}$$

unde:
- π reprezintă setul de task-uri;
- $Hi(\pi)$ este un subset de task-uri ce conține doar task-urile de criticalitate Hi.

- $[U_L, U_U]$: Utilizările sunt generate uniform din acest domeniu de valori, unde $0 \leq U_L \leq U_U \leq 1$.
- $[Z_L, Z_U]$: Raportul dintre utilizarea pentru nivelul de criticalitate Hi al unui task și utilizarea pentru nivelul de criticalitate Lo, unde $1 \leq Z_L \leq Z_U$.

Folosind acești parametri, algoritmul de generare a seturilor de task-uri (Algoritm 5) inițializează un set gol și adaugă task-uri până când limita superioară a utilizării totale este atinsă U_{bound} (linia 20 din pseudocod).

Algoritm 5. Generarea seturilor de task-uri.

Input: $[a, b], P_{Hi}, U_{bound}, [U_L, U_U], [Z_L, Z_U]$
Output: $\tau \in \{\tau_0, \tau_1, \dots, \tau_{n-1}\}$, unde n este numărul de task-uri

1 $\mathcal{U}(x, y)$ *este un număr generat uniform din domeniul de valori* $[x, y]$
2 $i \leftarrow 0$
3 $S_L \leftarrow 0$ ($valoarea\ curentă\ pentru\ U_{Lo}(\tau)$)
4 $S_H \leftarrow 0$ ($valoarea\ curentă\ pentru\ U_{Hi}(\tau)$)
5 **do**
6 $\quad u_H \leftarrow \mathcal{U}(U_L, U_U)$
7 $\quad u_L \leftarrow u_H / \mathcal{U}(Z_L, Z_U)$
8 $\quad T_i \leftarrow \mathcal{U}(a, b)$
9 $\quad L_i \leftarrow Hi\ with\ probability\ P_{Hi}; Lo\ with\ probability\ (1 - P_{Hi})$
10 $\quad$ **if** $L_i = Lo$ **then**
11 $\quad\quad U_{i,Lo} \leftarrow \min(u_L, U_{bound} - S_L)$
12 $\quad\quad S_L \leftarrow S_L + U_{i,Lo}$
13 $\quad$ **else**
14 $\quad\quad U_{i,Hi} \leftarrow \min(u_H, U_{bound} - S_H)$
15 $\quad\quad U_{i,Lo} \leftarrow \min(U_{i,Hi}, u_L, U_{bound} - S_L)$
16 $\quad\quad S_H \leftarrow S_H + U_{i,Hi}$
17 $\quad\quad S_L \leftarrow S_L + U_{i,Lo}$
18 $\quad$ **end if**
19 $\quad i \leftarrow i + 1$
20 **while** $max(S_L, S_H) < U_{bound}$

Pentru cazul în care utilizarea fiecărui task U_{i,L_j,PE_q}, este o matrice $p \times l$, unde p indică numărul de elemente de procesare din sistem, iar l reprezintă numărul de niveluri de criticalitate, metoda este modificată astfel încât să genereze $p \times l$ utilizări pentru fiecare task.

5.3 Funcțiile de bază ale mediului de simulare și evaluare

Funcțiile de bază ale mediului de simulare și evaluare sunt (Figura 10):

- **readTasks** – citește parametrii task-urilor din fișierul de la intrare (creat utilizând algoritmul pentru generarea seturilor de task-uri din MATLAB) și construiește o lista de structuri în care sunt adăugate toate task-urile din set. Fișierul de intrare va conține: numărul de elemente de procesare din sistem, numărul de moduri de rulare, numărul de task-uri din set și parametrii de timp ai fiecărui task.
- **checkProcessor** – selectează task-urile pe rând din listă pentru a fi testate pe fiecare element de procesare. În cazul în care testul este pozitiv, parametrul task-ului care salvează numărul elementului de procesare pe care urmează să fie planificat, este actualizat.
- **allocateTasks** – partiționează task-urile pe elementele de procesare corespunzătoare, conform funcției checkProcessor (parametrii task-urilor sunt adăugați în fișierul corespunzător fiecărui element de procesare).
- **executeAlg** – execută algoritmul de planificare la nivel de element de procesare. Ca parametru de intrare, funcția primește numărul elementului de procesare pe care urmează să ruleze algoritmul de planificare respectiv. Planificarea finală este stocată în mai multe fișiere, câte unul pentru fiecare element de procesare și mod de rulare.

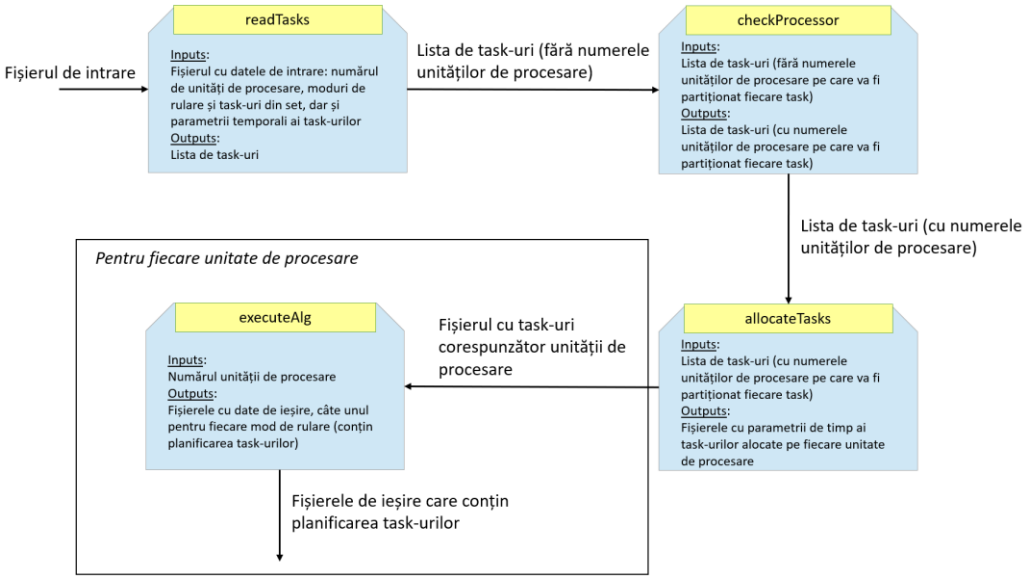


Figura 10. Funcțiile de bază ale mediului de simulare și evaluare.

5.4 Preluarea rezultatelor și interfața grafică

Interfața grafică pentru vizualizarea planificării task-urilor a fost dezvoltată în MATLAB. Planificarea task-urilor (momentele de începere și finalizare a execuției fiecărui task) este preluată din fișierele de la ieșire pentru toate elementele de

procesare și afișată sub forma unei axe cu blocuri de dimensiuni și culori diferite. Fiecărui task îi este asociată o anumită culoare, în timp ce dimensiunea blocului este dată de timpul de execuție alocat de către sistem. Lungimea axei este egală cu cel mai mare divizor comun al perioadelor task-urilor de pe elementul de procesare respectiv. Interfața grafică pentru vizualizarea planificării, utilizând un algoritm oarecare, în cazul unui set cu 5 task-uri este ilustrată în Figura 11:

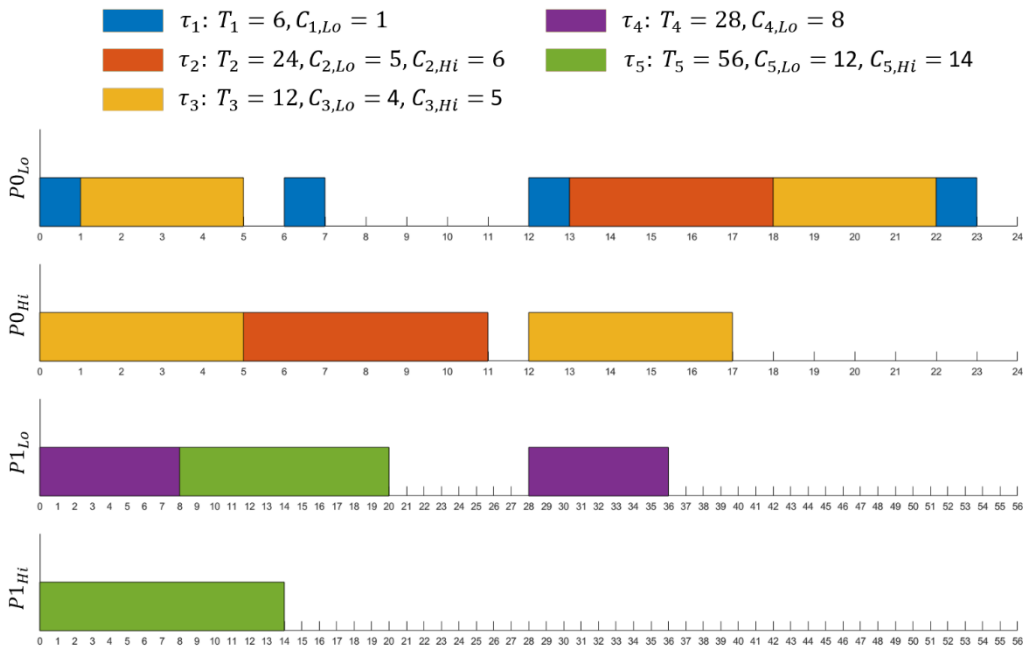


Figura 11. Interfața grafică pentru vizualizarea planificării.

6 EVALUAREA PERFORMANȚEI

O serie de simulări experimentale au fost efectuate pentru a evalua eficiența tehnicii de mapare Best Affinity Fit (BAF). Astfel, s-a realizat compararea algoritmului cu două metode de mapare din literatura de specialitate, Best Fit Decreasing Utilization (BFDU) și Best Fit Decreasing Criticality (BFDC) [191-193]. Tehnica BAF alocă task-uri pe elementele de procesare în funcție de valoarea scorului de afinitate, în timp ce pentru BFDU și BFDC, task-urile sunt mai întâi ordonate descrescător în funcție de utilizare, respectiv, în funcție de nivelul de criticalitate, iar apoi partiționate pe fiecare elemente de procesare; elementele de procesare sunt, de asemenea, ordonate descrescător în funcție de utilizare. În plus, două strategii de atribuire a scorului de afinitate au fost evaluate: o metodă setează afinitatea în funcție de WCET, iar cealaltă în funcție de nivelul de criticalitate.

S-a considerat un sistem cu două niveluri de criticalitate $\{Lo, Hi\}$. Parametrii fiecărui task τ_i sunt generați după cum urmează:

- Nivelul de criticalitate: $L_i = Hi$ cu o probabilitate de P_{Hi}, altfel $L_i = Lo$.
- Perioada: T_i este o valoare generată aleatoriu uniform în intervalul $[10, 100]$.
- Deadline-ul este egal cu perioada $D_i = T_i$.
- Utilizarea fiecărui task U_{i,L_j,PE_q}, este o matrice $p \times l$, unde p reprezintă numărul de elemente de procesare din sistem, iar l indică numărul de niveluri de criticalitate.
- WCET-ul pentru nivelul de criticalitate Lo: $C_{i,L_{Lo}PE_q} = U_{i,L_{Lo}PE_q} \cdot T_i$.
- WCET-ul pentru nivelul de criticalitate Hi: $C_{i,L_{Hi}PE_q} = U_{i,L_{Hi}PE_q} \cdot T_i$ dacă $L_i = Hi$. Altfel, $C_{i,L_{Hi}PE_q} = C_{i,L_{Lo}PE_q}$.
- Scorul de afinitate este atribuit folosind una din cele două metode descrise în Subcapitolul 4.2.

Parametrii utilizați pentru generarea setului de task-uri sunt definiți în legenda figurilor. Pentru fiecare reprezentare grafică, un singur parametru variază, iar ceilalți rămân ficși. Utilizarea pe fiecare element de procesare trebuie să îndeplinească condiția de necesitate legată de metrica de încărcare pentru orice sistem cu m elemente de procesare (4-7). Fiecare punct de date de pe grafic a fost determinat prin generarea a 1000 de seturi de task-uri.

6.1 BAF vs. BFDU

Valorile scorului de afinitate au fost atribuite în funcție de WCET-ul din modul de criticalitate Hi pentru task-urile de criticalitate Hi, și în funcție de WCET-ul din modul de criticalitate Lo pentru task-urile de criticalitate Lo.

În Figura 12 limita superioară a utilizării totale pentru fiecare set de task-uri (axa x) variază de la 0.4 la 1.0 cu un pas de 0.1, valoare înmulțită apoi cu numărul de elemente de procesare din sistem. Pentru Figura 13 numărul de elemente de procesare (axa x) variază de la 2 la 12 cu un pas de 2. În Figura 14 numărul de task-uri din set (axa x) variază de la 10 la 24 cu un pas de 3, iar Figura 15 ilustrează variația procentului de task-uri de criticalitate Hi dintr-un set (axa x) de la 0.2 la 0.6 cu un pas de 0.1. Utilizarea totală medie este definită pe axa y în toate figurile.

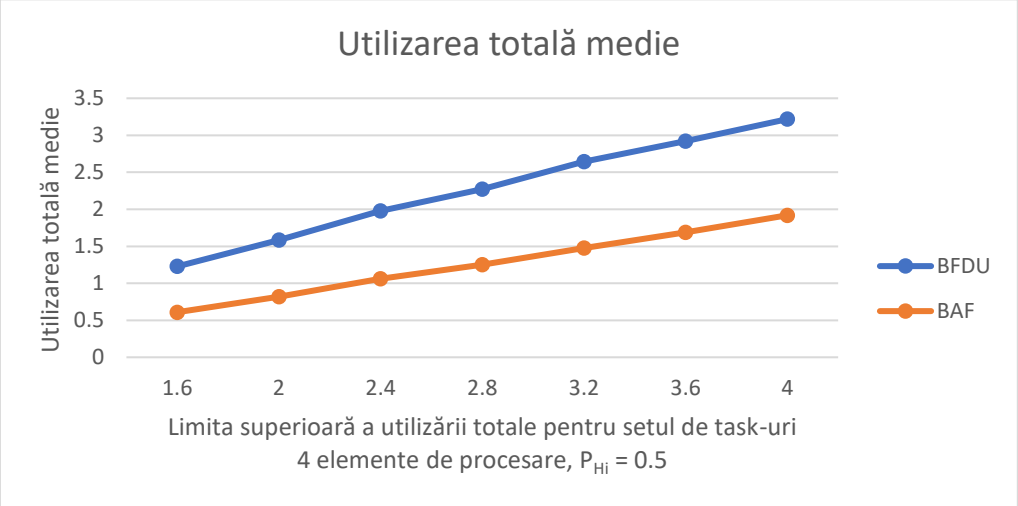


Figura 12. Utilizarea totală medie a elementelor de procesare, obținută prin variația limitei superioare a utilizării totale pentru fiecare set de task-uri.
$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

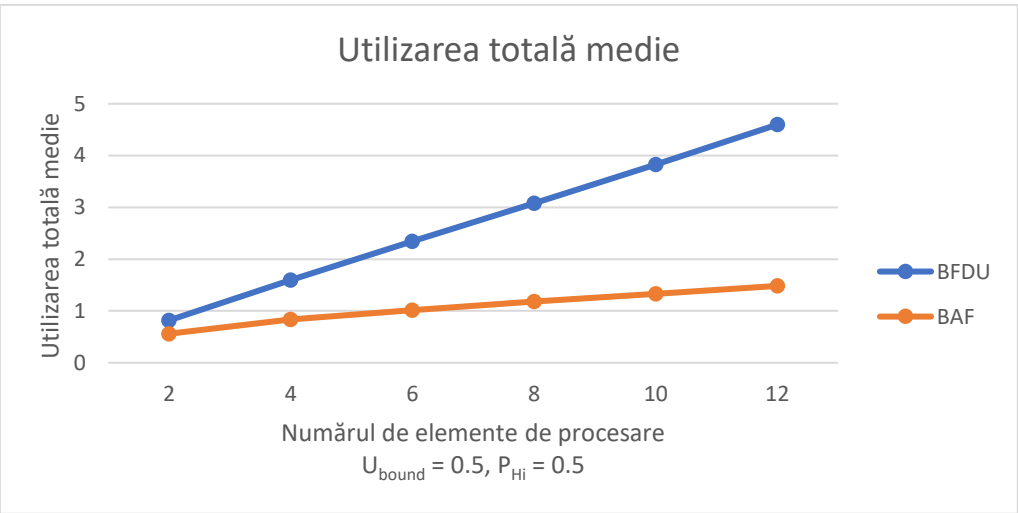


Figura 13. Utilizarea totală medie a elementelor de procesare, obținută prin variația numărul de elemente de procesare din sistem.
$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

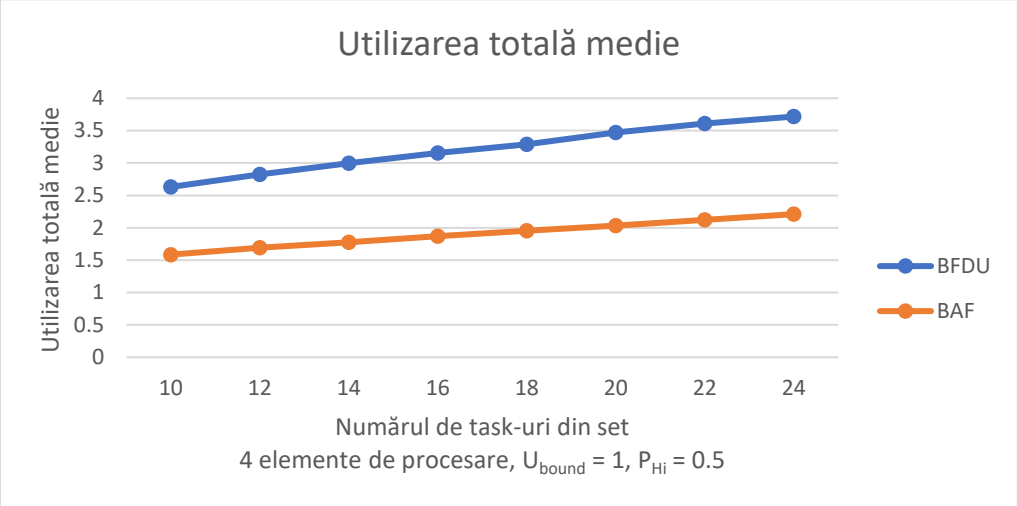


Figura 14. Utilizarea totală medie a elementelor de procesare, obținută prin variația numărului de task-uri din fiecare set.
$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

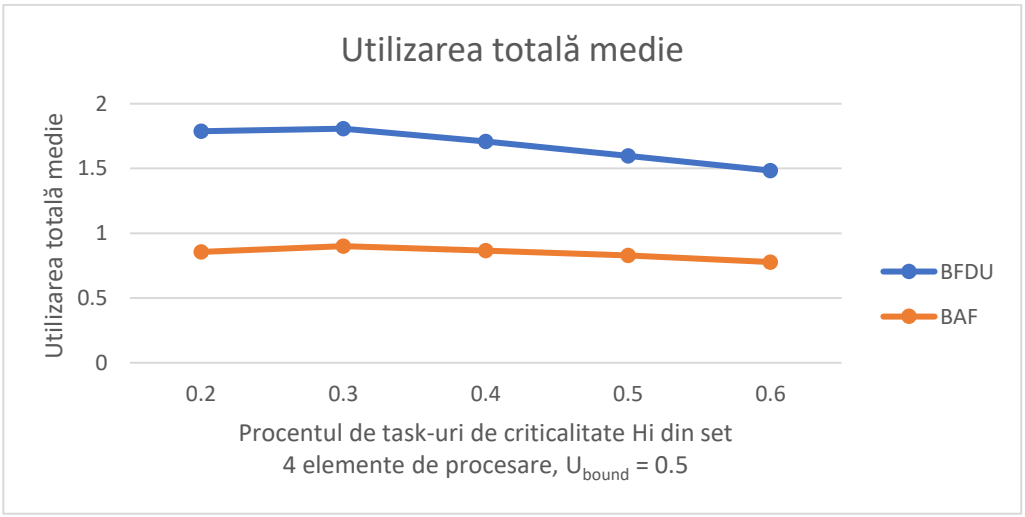


Figura 15. Utilizarea totală medie a elementelor de procesare, obținută prin variația procentului de task-uri de criticalitate Hi din fiecare set.
$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

6.2 BAF vs. BFDC

În acest caz, valorile scorului de afinitate au fost atribuite în funcție de nivelul de criticalitate al fiecărui task. Au fost realizare patru experimente, asemănătoare

primului set de simulări. Fiecare punct de date de pe grafic a fost determinat prin generarea aleatorie a 1000 de seturi de task-uri.

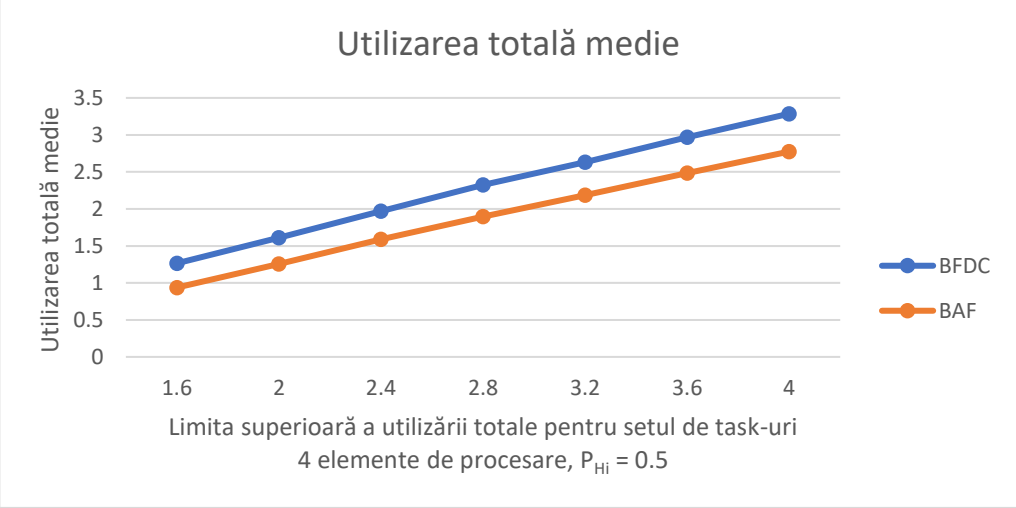


Figura 16. Utilizarea totală medie a elementelor de procesare, obținută prin variația limitei superioare a utilizării totale pentru fiecare set de task-uri.
$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

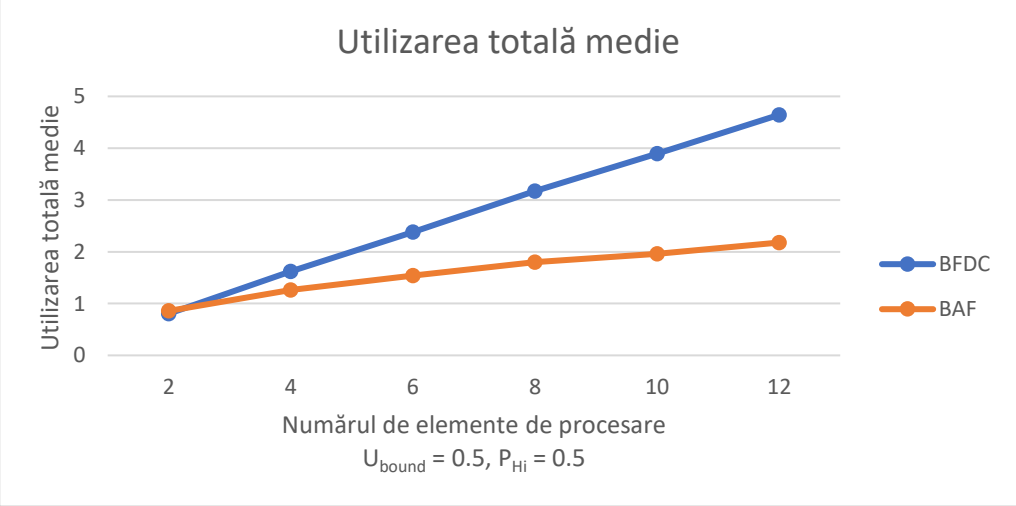


Figura 17. Utilizarea totală medie a elementelor de procesare, obținută prin variația numărul de elemente de procesare din sistem.
$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

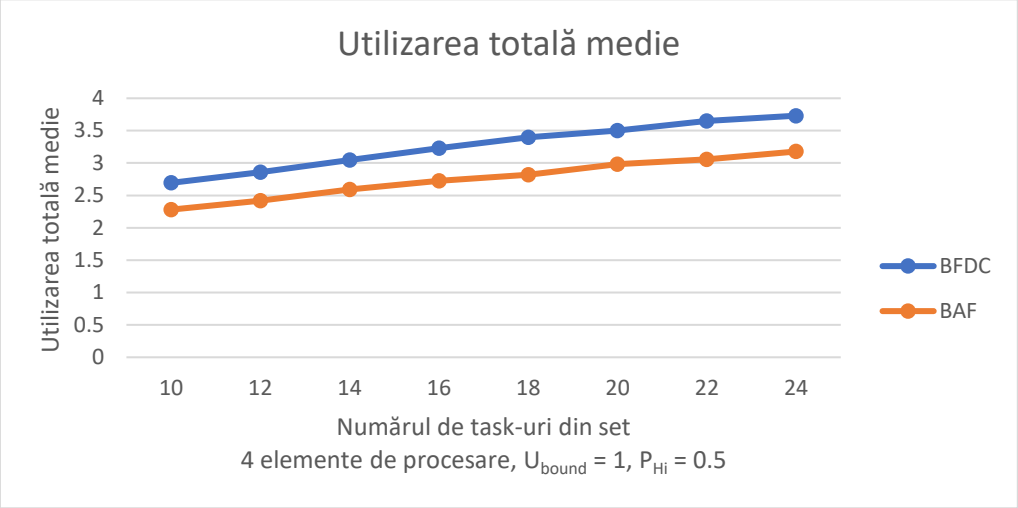


Figura 18. Utilizarea totală medie a elementelor de procesare, obținută prin variația numărului de task-uri din fiecare set.
$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

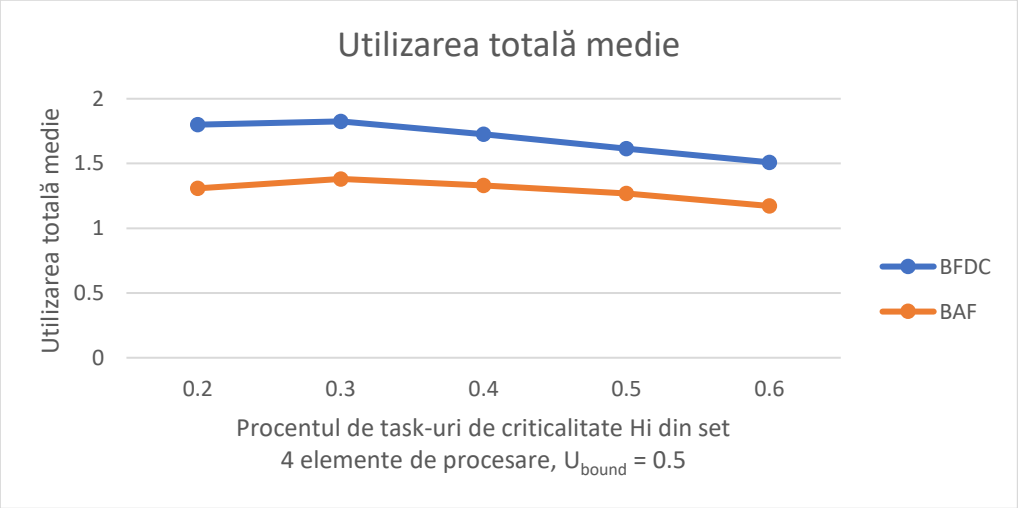


Figura 19. Utilizarea totală medie a elementelor de procesare, obținută prin variația procentului de task-uri de criticalitate Hi din fiecare set.
$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

Deviația medie a scorului de afinitate pentru seturile de task-uri generate în Figura 16 este reprezentată în Figura 20, respectând formula (4-10). Valorile obținute sunt, de asemenea, afișate și în Tabel 11.

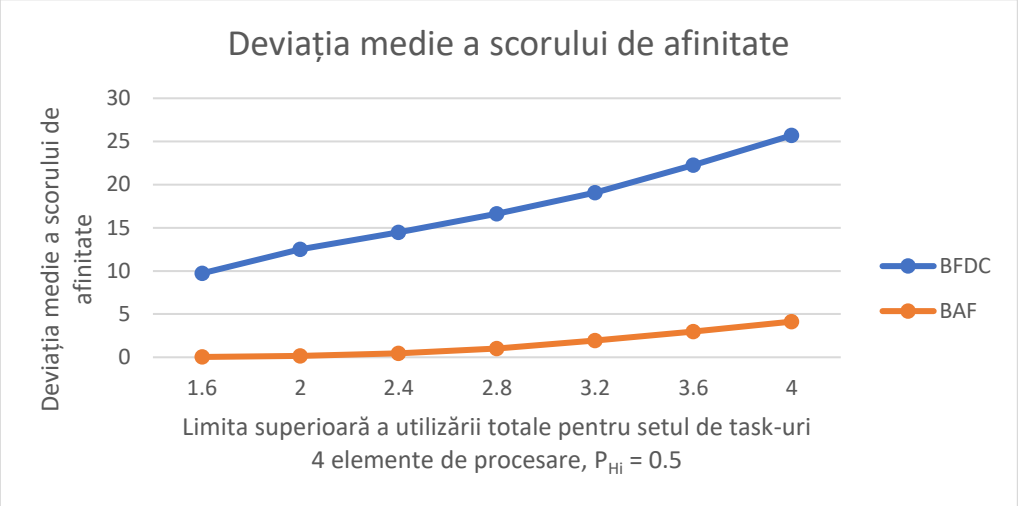


Figura 20. Deviația medie a scorului de afinitate, obținută prin variația limitei superioare a utilizării totale pentru fiecare set de task-uri.
$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

Tabel 11. Valorile pentru deviația medie a scorului de afinitate din Figura 20.

Limita superioară a utilizării totale pentru setul de task-uri	**Deviația medie a scorului de afinitate pentru BAF**	**Deviația medie a scorului de afinitate pentru BFDC**
1.6	0.039	9.727
2	0.144	12.518
2.4	0.473	14.484
2.8	1.011	16.613
3.2	1.926	19.064
3.6	2.992	22.261
4	4.130	25.698

7 FENP_MC: FIXED EXECUTION NON-PREEMPTIVE MIXED CRITICALITY

În această secțiune este prezentat un algoritm non-preemptiv pentru MCSs, care rulează într-un mediu time-triggered. Acesta este introdus ca răspuns cerințelor pentru o execuție fără jitter, cu aplicabilitate asupra task-urilor utilizate în domenii precum: procesarea semnalelor, diferite tipuri de sincronizări, bucle de control, etc. [21, 26, 194, 195].

7.1 Modelarea task-urilor perfect periodice

Modelele de execuție pentru task-uri perfect periodice în sisteme de timp real cu niveluri mixte de criticalitate se bazează pe modelul propus de Liu și Layland în 1973 [196]. Acest model impune un comportament periodic doar din punctul de vedere al timpului de activare pentru fiecare job. În majoritatea sistemelor bazate pe modelul lui Liu și Layland, timpul de începere a execuției este pseudo-periodic [197].

Un alt model, foarte asemănător cu cel propus de Liu și Layland, dar care abordează cazul special de task-uri periodice în timp real, a fost introdus în [195]. Aici, task-urile sunt denumite FModXs (Fixed Execution Executable Modules). Pornind de la această versiune, teza de față propune un model de execuție pentru task-uri perfect periodice în sisteme de timp real:

$$M_i = \{T_i, D_i, C_i, S_i\} \tag{7-1}$$

unde:

- T_i reprezintă perioada task-ului periodic i;
- D_i este momentul până la care trebuie încetată execuția unei instanțe relativ la timpul de activare;
- C_i reprezintă timpul de execuție;
- S_i oferă timpul de începere (start) a execuției, relativ la momentul de activare.

Pe baza abordării lui Vestal, care extinde modelul lui Liu și Layland în domeniul sistemelor cu niveluri mixte de criticalitate, această teză propune următorul model pentru task-uri perfect periodice în MCSs:

$$M_i = \left\{T_i, D_i, L_i, \{C_{i,L_j} | \, j \in 1 \dots l\}, \{S_{i,L_j} | \, j \in 1 \dots l\}\right\} \tag{7-2}$$

unde:

- M_i este un task cu execuție fixă, de criticalitate mixtă, MC-FModX (Mixed Criticality Fixed Execution Executable Module);
- l reprezintă numărul de niveluri de criticalitate;
- T_i este perioada task-ului periodic i;
- D_i indică momentul până la care trebuie încetată execuția unei instanțe, relativ la timpul de activare;
- L_i semnifică nivelul de criticalitate (1 fiind nivelul cel mai scăzut);

- C_{i,L_j} este timpul de execuție (vector de valori – câte o valoare pentru fiecare nivel de criticalitate mai mic sau egal cu L_j, exprimând timpul de execuție pentru cazul cel mai defavorabil pentru fiecare nivel de criticalitate);
- S_{i,L_j} reprezintă timpul de start (vector de valori – câte o valoare pentru fiecare nivel de criticalitate mai mic sau egal cu L_j, indicând timpul de începere a execuției, relativ la momentul de activare).

Un task constă dintr-o serie de job-uri, fiecare job moștenind setul de parametrii al task-ului, (T_i, L_i, D_i), la care se adaugă proprii parametri [54]. Astfel, job-ul k al task-ului M_i este caracterizat de:

$$J_{i,k} = \{a_{i,k}, d_{i,k}, c_{i,k}, s_{i,k}, T_i, D_i, L_i\} \tag{7-3}$$

unde:

- $a_{i,k}$ este timpul de activare ($a_{i,k+1} - a_{i,k} \geq T_i$);
- $d_{i,k}$ indică deadline-ul absolut ($d_{i,k} = a_{i,k} + D_i$);
- $c_{i,k}$ reprezintă timpul de execuție alocat de către sistem, care este dependent de modul de rulare al sistemului (pentru L_j, $c_{i,k} = C_{i,L_j}$);
- $s_{i,k}$ oferă timpul absolut de începere a execuției pentru job-ul k al task-ului i, care este, de asemenea, dependent de modul de rulare al sistemului;
- T_i, D_i, L_i au aceeași semnificație ca în (7-2).

7.2 Planificarea task-urilor perfect periodice

Definiția 2. *Execuția task-ului i este perfect periodică dacă pentru fiecare job k al task-ului i, $J_{i,k}$, diferența dintre timpul absolut de începere a execuției pentru job-urile k și $k-1$ este constantă:*

$$s_{i,1} - s_{i,0} = s_{i,2} - s_{i,1} = \cdots = s_{i,n} - s_{i,n-1} = T_i \tag{7-4}$$

Pentru a exemplifica execuția unui algoritm perfect periodic, se consideră setul de task-uri prezentat în Tabel 12, care va fi planificat pe un sistem cu o singură unitate de procesare, cu două niveluri de criticalitate (Hi – criticalitate ridicată și Lo – criticalitate scăzută). În Tabel 12, T_i este perioada task-ului i, D_i reprezintă deadline-ul, L_i indică nivelul de criticalitate, $C_{i,L_{Lo}}$ reprezintă timpul de execuție pentru modul de criticalitate Lo, iar $C_{i,L_{Hi}}$ este timpul de execuție pentru modul de criticalitate Hi.

Tabel 12. Exemplu de set cu trei task-uri.

Task	T_i	D_i	L_i	$C_{i,L_{Lo}}$	$C_{i,L_{Hi}}$
M_1	10	10	Lo	3	-
M_2	20	20	Hi	2	4
M_3	30	30	Hi	5	6

Figura 21 ilustrează timpii de start ai task-urilor pentru modul de criticalitate Lo.

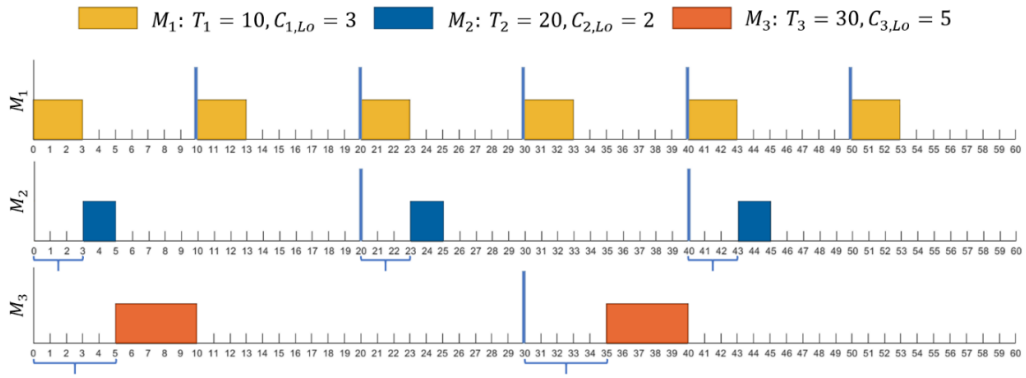


Figura 21. Timpii de start pentru exemplul de set cu trei task-uri din Tabel 12 în modul de criticalitate Lo.

Într-o platformă cu niveluri mixte de criticalitate, relația (7-4) trebuie să fie respectată pentru toate modurile de rulare ale sistemului (adică pentru toate nivelurile de criticalitate), după cum se poate observa în Figura 22. Aici $P0_{Lo}$ reprezintă nivelul de criticalitate Lo (Figura 22 a.), iar $P0_{Hi}$ indică nivelul de criticalitate Hi (Figura 22 b.).

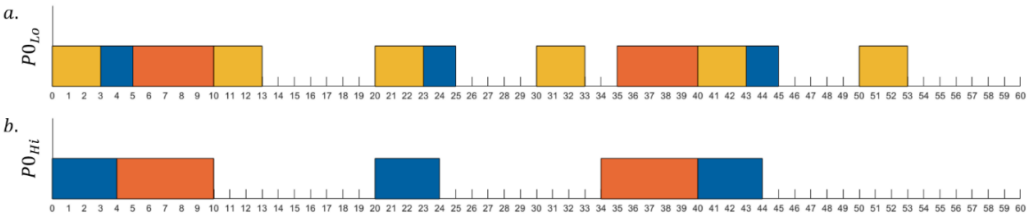


Figura 22. Planificarea exemplului de set cu trei task-uri din Tabel 12 în a. modul de criticalitate Lo și b. modul de criticalitate Hi.

În MCSs, task-uri cu cerințe de criticalitate diferite împart aceeași platformă hardware. Astfel, consecințele pentru depășirea unui deadline variază ca severitate în funcție de fiecare task [30]. Pentru a proteja task-urile critice de interferența task-urilor mai puțin critice, fiecărui task îi este atribuit un nivel de criticalitate, existând astfel mai multe niveluri de garanție pentru task-uri care rulează în diferite scenarii de rulare, numite moduri de criticalitate.

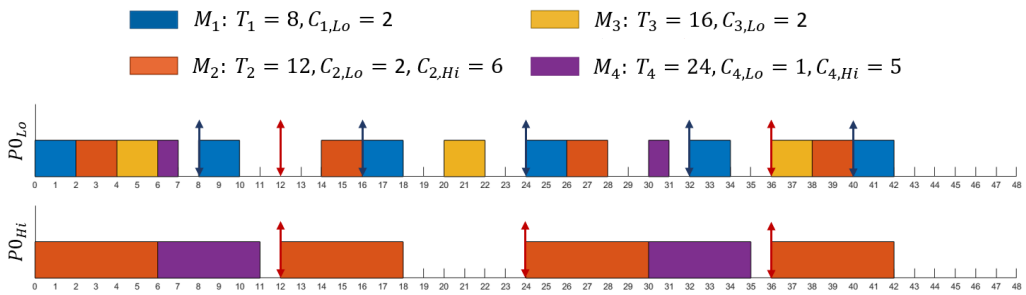
După cum s-a specificat și în [31] pentru modelul clasic de task-uri din MCSs, sistemul își începe execuția în modul de criticalitate cel mai scăzut. Dacă toate job-urile respectă nivelul de garanție impus de acest mod, atunci sistemul va rămâne în modul de rulare respectiv. Pe de altă parte, dacă execuția acestora depășește nivelul de garanție impus, atunci va avea loc o schimbare a modului de rulare (mode-change), iar sistemul va rula conform unui nivel de garanție mai ridicat.

În continuare este ilustrat un exemplu în care FENP_MC tratează situația apariției unui mode-change. Se va considera setul de task-uri prezentat în Tabel 13:

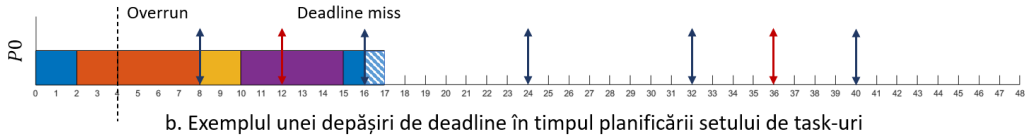
Tabel 13. Exemplu de set cu patru task-uri.

Task	T_i	D_i	L_i	$C_{i,L_{Lo}}$	$C_{i,L_{Hi}}$
M_1	8	8	Lo	2	-
M_2	12	12	Hi	2	6
M_3	16	16	Lo	2	-
M_4	24	24	Hi	1	5

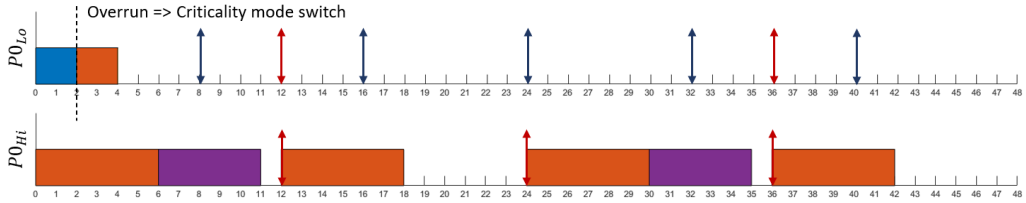
În Figura 23 a. sunt oferite două tabele ale planificării ($P0_{Lo}$ pentru modul de criticalitate Lo și $P0_{Hi}$ pentru modul de criticalitate Hi, unde $Lo < Hi$). Sistemul începe execuția în modul de criticalitate Lo, utilizând tabelul planificării $P0_{Lo}$. La momentul de timp 4, task-ul M_2 depășește bugetul de timp alocat pentru modul de criticalitate Lo, ceea ce cauzează o schimbare a modului de criticalitate (criticality mode switch – Figura 23 b.). Sistemul își continuă execuția conform tabelului planificării $P0_{Hi}$, începând cu momentul de timp 0. În modul de rulare Hi, toate task-urile de criticalitate Lo sunt abandonate și doar task-urile de criticalitate Hi sunt planificate conform garanției timpului de execuție pentru nivelul de criticalitate Hi (Figura 23 c.).



a. Planificarea exemplului de set cu patru task-uri în modurile de criticalitate Lo și Hi.



b. Exemplul unei depășiri de deadline în timpul planificării setului de task-uri folosind Fixed Execution Non-Preemptive (FENP).



c. Exemplu de schimbare a modului de criticalitate în timpul planificării setului de task-uri utilizând Fixed Execution Non-Preemptive Mixed Criticality (FENP_MC).

Figura 23. Exemplu de schimbare a modului de criticalitate.

7.3 Analiza fezabilității

7.3.1 Aspecte teoretice

În continuare este prezentat un test de fezabilitate exact, pentru execuția perfect periodică a task-urilor într-un context non-preemptiv. Acest tip de execuție poartă numele de Fixed Execution Non-Preemptive (FENP). Testul este analog cu cel utilizat în [195].

Se consideră un set $M = \{M_1, M_2, \dots, M_n\}$ de n task-uri independente cu execuție fixă, de criticalitate mixtă (MC-FModXs), ordonate crescător în funcție de perioadă. Task-urile sunt caracterizate de aceeași parametri, precum cei descriși în expresia (7-2), astfel:

$$M_i \equiv \left\{T_i, D_i, L_i, \{C_{i,L_j} | \, j \in 1 \dots l\}, \{S_{i,L_j} | \, j \in 1 \dots l\}\right\}, \quad \text{unde pentru orice task } k,\ T_k \leq T_i \text{ pentru } k < i \tag{7-5}$$

Definiția 3. *Setul de task-uri M este FENP planificabil într-un sistem cu niveluri mixte de criticalitate, dacă și numai dacă, setul de task-uri M este FENP planificabil pentru orice nivel de criticalitate L_j, unde $j \in 1 \dots l$.*

Definiția 4. *Setul de task-uri M este FENP planificabil într-un sistem cu niveluri mixte de criticalitate pentru un nivel de criticalitate L_j dacă toate task-urile din setul M de criticalitate egală sau mai mare decât L_j sunt FENP planificabile utilizând testul de fezabilitate de mai jos. Doar parametrii pentru nivelul L_j (C_{i,L_j} și S_{i,L_j}) sunt considerați în acest caz.*

Testele de fezabilitate sunt bazate pe o funcție de mapare a execuției, definită în continuare.

Definiția 5. *O mapare cu execuție fixă a task-ului M_k peste perioada task-ului M_i este o funcție de forma:*

$$\Delta_{M_i/M_k}: \{0, 1, \dots, T_i - 1\} \to \{0, 1\}$$
$$\Delta_{M_i/M_k}(\tau) = \bigcup_{x=0}^{\frac{1}{GCD(T_k,T_i)} \cdot T_k - 1} M_k(\tau + x \cdot T_i) \tag{7-6}$$

unde:

- *τ reprezintă o funcție discretă de timp cu valori cuprinse între 0 și T_i;*
- *$GCD(T_k, T_i)$ calculează cel mai mare divizor comun al perioadelor task-urilor M_k și M_i;*
- *$M_k(\tau + x \cdot T_i)$ este funcția de execuție a task-ului M_k:*

$$M_k: \boldsymbol{N} \to \{0, 1\},$$
$$M_k(t) = \sigma(t \bmod T_k - S_k) - \sigma(t \bmod T_k - S_k - C_k) \tag{7-7}$$

unde:

- mod *este operația modulo;*
- σ *este funcția treaptă unitate:*

$$\sigma: \boldsymbol{Z} \to \{0,1\}, \sigma(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases} \tag{7-8}$$

Pentru un anumit nivel de criticalitate L_j, relația (7-7) devine:

$$\begin{gathered} M_k: \boldsymbol{N} \to \{0,1\}, \\ M_k(t) = \sigma\left(t \bmod T_k - S_{k,L_j}\right) - \sigma\left(t \bmod T_k - S_{k,L_j} - C_{k,L_j}\right) \end{gathered} \tag{7-9}$$

Test de fezabilitate: Pentru un nivel de criticalitate dat L_j, un subset de task-uri M_{L_j} de criticalitate $L_i \geq L_j$, este planificabil dacă și numai dacă:

$$\begin{gathered} \forall M_i \in M_{L_j}, M_i \equiv \left\{T_i, D_i, L_i, C_{i,L_j}, S_{i,L_j}\right\}, \\ \exists t_i \in \left\{0, 1, \dots, T_i - C_{i,L_j}\right\} \text{ astfel încât } \bigcup_{\tau=t_0}^{t_i+C_{i,L_j}-1} \bigcup_{k=1}^{i-1} \Delta_{M_i/M_k}(\tau) = 0 \end{gathered} \tag{7-10}$$

unde:

- t_i este o instanță în timp discret între 0 și cel mai târziu moment de start posibil al task-ului M_i;
- $\Delta_{M_i/M_k}(\tau)$ este definit de relația (7-6).

7.3.2 Exemple de execuție

Pentru o mai bună înțelegere, Figura 24 ilustrează funcția de mapare a execuției într-un sistem cu două niveluri de criticalitate pentru setul de task-uri prezentat în Tabel 12.

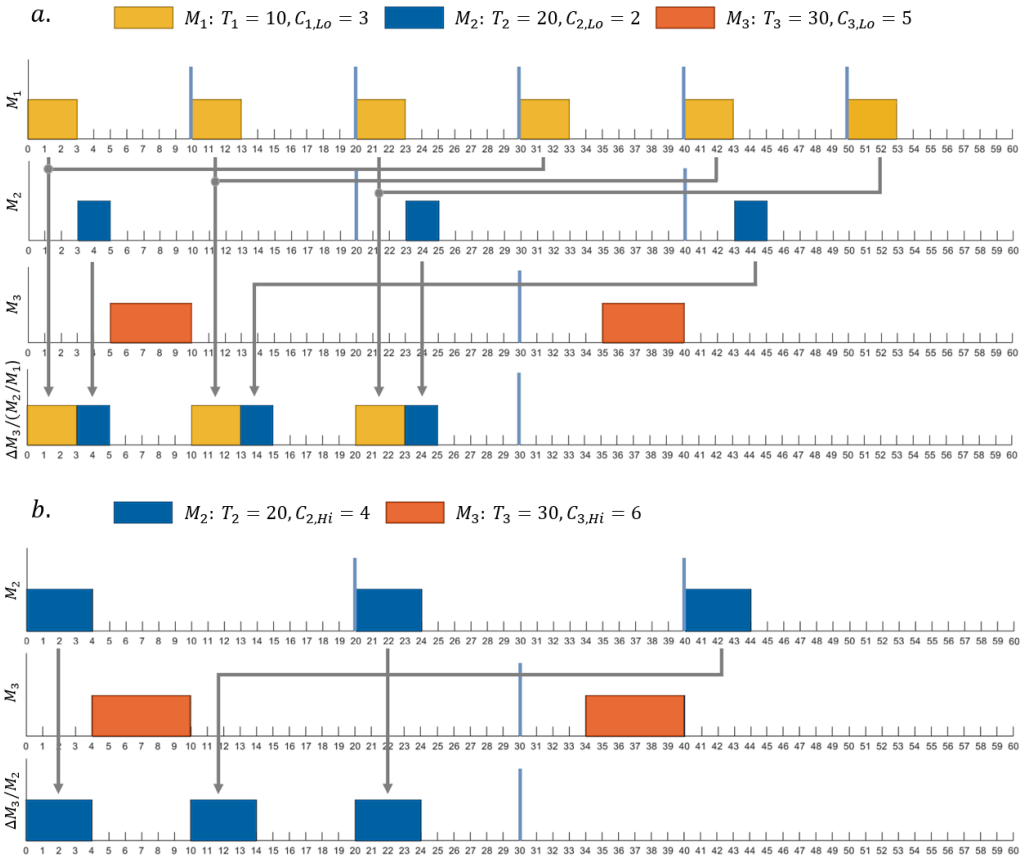


Figura 24. Funcția de mapare a execuției pentru exemplul de set cu trei task-uri din Tabel 12 în a. modul de criticalitate Lo și b. modul de criticalitate Hi.

7.4 Implementare

Testul de fezabilitate este prezentat sub formă de pseudocod în Algoritm 6. Acesta este o adaptare pentru sisteme cu niveluri mixte de criticalitate a testului de fezabilitate din [195].

Algoritm 6. Test de fezabilitate.

Input: Γ_q (tabelul planificării pentru unitatea de procesare q), $critLevel$ (modul de criticalitate al sistemului)
Output: $FAILURE$ pentru un test de fezabilitate negativ, $SUCCESS$ altfel

1 *sortează* Γ_q *în funcție de* T_i *în ordine crescătoare*
2 **for** $i \in \{1, 2, \dots, size\ of\ \Gamma_q - 1\}$ **do**
3 **for** $j \in \{0, 1, \dots, i-1\}$ **do**

```
4          gcd ← cel mai mare divizor comun al lui T_i şi T_j
5          if critLevel = Lo and C_{i,critLevel} + C_{j,critLevel} > gcd then
6              return FAILURE
7          end if
8          if critLevel = Hi and L_i = Hi and L_j = Hi and C_{i,critLevel} +
       C_{j,critLevel} > gcd then
9              return FAILURE
10         end if
11     end for
12 end for
13 return SUCCESS
```

7.5 FENP_MC

7.5.1 Aspecte teoretice

În timp ce algoritmii de planificare event-driven garantează doar o execuție pseudo-periodică, abordările time-triggered pot să ofere o soluție pentru task-uri perfect periodice, dacă se ignoră valoarea jitter-ului introdus de schimbarea modului de criticalitate.

În continuare este propusă o adaptare pentru MCSs a algoritmului de planificare în timp real, table-driven, FENP [195] pentru sisteme cu o singură unitate de procesare, precum și a variantei sale partiționată, P_FENP [76] pentru sisteme cu mai multe unități de procesare.

Algoritmul Fixed Execution Non-Preemptive (FENP) a fost conceput cu scopul de a oferi predictibilitate maximă pentru execuția de task-uri perfect periodice (FModXs) într-un context non-preemptiv.

Deoarece algoritmul FENP respectă expresia (7-4), fiecare timp de start corespunzător job-ului $J_{i,k}$, al task-ului i, poate fi determinat cunoscând momentul de start al job-ului anterior $J_{i,k-1}$:

$$s_{i,k} = s_{i,k-1} + T_i \tag{7-11}$$

Mai mult decât atât, $s_{i,k}$ poate fi determinat static, în mod direct:

$$s_{i,k} = s_{i,0} + k \cdot T_i \tag{7-12}$$

Prin dezvoltarea unei metode de planificare statică pe baza relațiilor (7-11) și (7-12), se va obține o execuție a task-urilor fără jitter.

Algoritmul de planificare FENP_MC creează, în etapa offline, un tabel al planificării pentru fiecare nivel de criticalitate al sistemului pe baza expresiei (7-11) și a testului de fezabilitate propus în [195] pentru sisteme de timp real, și care a fost adaptat și prezentat în următoarea secțiune pentru sisteme cu niveluri mixte de criticalitate.

Tabelul planificării este reprezentat printr-un tablou de structuri de forma:

$$\Gamma_q = \{TaskID;\ StartTime\} \tag{7-13}$$

unde Γ_q este sortat în ordine crescătoare pentru fiecare perioadă de planificare în funcție de timpul de start pentru fiecare job din sistem.

7.5.2 Exemple de execuție

Tabelele planificării pentru modurile de criticalitate Lo și Hi în cazul setului de trei task-uri din Tabel 12 sunt prezentate în Tabel 14, respectiv în Tabel 15:

Tabel 14. Tabelul planificării pentru modul de criticalitate Lo în cazul exemplul de set cu trei task-uri din Tabel 12.

TaskID	StartTime
1	0
2	3
3	5

Tabel 15. Tabelul planificării pentru modul de criticalitate Hi în cazul exemplul de set cu trei task-uri din Tabel 12.

TaskID	StartTime
2	0
3	4

În mod asemănător, tabelele planificării pentru setul de patru task-uri din Tabel 13 sunt evidențiate în Tabel 16 și Tabel 17:

Tabel 16. Tabelul planificării pentru modul de criticalitate Lo în cazul exemplul de set cu patru task-uri din Tabel 13.

TaskID	StartTime
1	0
2	2
3	4
4	6

Tabel 17. Tabelul planificării pentru modul de criticalitate Hi în cazul exemplul de set cu patru task-uri din Tabel 13.

TaskID	StartTime
2	0
4	6

Pentru un exemplu de set cu șapte task-uri (Tabel 18), tabelele planificării sunt prezentare în Tabel 19 și Tabel 20:

Tabel 18. Exemplu de set cu șapte task-uri.

Task	T_i	D_i	L_i	$C_{i,L_{Lo}}$	$C_{i,L_{Hi}}$
M_1	9	9	Lo	1	-
M_2	27	27	Hi	1	3
M_3	36	36	Hi	1	2
M_4	45	45	Lo	2	-

M_5	63	63	Hi	2	3
M_6	72	72	Lo	2	-
M_7	90	90	Lo	3	-

Tabel 19. Tabelul planificării pentru modul de criticalitate Lo în cazul exemplul de set cu șapte task-uri din Tabel 18.

TaskID	StartTime
1	0
2	1
3	2
4	3
5	5
6	7
7	11

Tabel 20. Tabelul planificării pentru modul de criticalitate Hi în cazul exemplul de set cu șapte task-uri din Tabel 18.

TaskID	StartTime
2	0
3	3
5	5

7.5.3 Implementare

Funcția de mapare a execuției, prezentată în Algoritm 7 este utilizată de către funcția de calculare a timpilor de start, din Algoritm 8. Ambii algoritmi sunt adaptați pentru sisteme cu niveluri mixte de criticalitate pe baza metodelor introduse în [195].

Algoritm 7. MFunc.

Input: i (indexul task-ului), t (instanța de timp), $critLevel$ (modul de criticalitate al sistemului)
Output: $\Delta sigma = sigma1 - sigma2$

```
temp ← mod(t, T_i) − S_{i,critLevel}
if temp < 0 then
    sigma1 ← 0
else
    sigma1 ← 1
end if
if temp − C_{i,critLevel} < 0 then
    sigma2 ← 0
else
    sigma2 ← 1
end if
return sigma1 − sigma2
```

Timpii de start pentru instanțele task-urilor sunt calculați folosind Algoritm 8 și salvați în cele două tabele ale planificării (Lo și Hi) pentru fiecare unitate de

procesare din sistem. Tabelele vor fi sortate în ordine crescătoare în funcție de timpii de start ai job-urilor.

Algoritm 8. Calculare timpi de start.

Input: Γ_q (tabelul planificării pentru unitatea de procesare q),
$critLevel$ (modul de criticalitate al sistemului)
Output: $FAILURE$ dacă timpii de start nu pot fi calculați, $SUCCESS$ altfel

```
1   for i ∈ {1, 2, ..., size of Γ_q − 1} do
2       schedulable ← 0
3       count ← 0
4       gcd ← 1
5       StartTime ← −1
6       for t ∈ {0, 1, ..., T_i} do
7           delta ← 0
8           for j ∈ {0, 1, ..., i − 1} do
9               if delta ≠ 0 then
10                  break
11              end if
12              gcd ← cel mai mare divizor comun al lui T_i şi T_j
13              for k ∈ {0, 1, ..., T_j/gcd − 1} do
14                  if delta ≠ 0 or MFunc(j, t + k * T_i, critLevel) ≠ 0 then
15                      delta ← 1
16                      break
17                  end if
18              end for
19          end for
20          if count ≥ C_{j,critLevel} then
21              S_{i,critLevel} ← StartTime
22              schedulable ← 1
23              break
24          end if
25          if delta ≠ 0 then
26              count ← 0
27              StartTime ← −1
28          else
29              count ← count + 1
30              if StartTime = −1 then
31                  StartTime ← t
32              end if
33          end if
34      end for
35      if StartTime = −1 then
36          return FAILURE
37      end if
38      if schedulable = 0 then
39          return FAILURE
```

```
    end if
end for
return SUCCESS
```

8 P_FENP_MC: PARTITIONED FIXED EXECUTION NON-PREEMPTIVE MIXED CRITICALITY

8.1 Aspecte teoretice

Această teză propune o adaptare a metodei P_FENP pentru sisteme cu mai multe unități de procesare, cu niveluri mixte de criticalitate, denumită P_FENP_MC. Algoritmul de mapare este asemănător cu cel propus în [76].

P_FENP_MC constă din două etape, o etapă offline și una online. Partiționarea task-urilor pe fiecare unitate de procesare se realizează offline. Testul de fezabilitate este rulat pe fiecare unitate de procesare, urmat de crearea tabelului planificării pentru unitatea de procesare respectivă. În etapa online task-urile vor fi planificate conform tabelelor planificării. Inițial, sistemul rulează în modul de criticalitate Lo, deci task-urile sunt planificate în funcție de tabelul planificării pentru modul de criticalitate Lo. Dacă un job depășește valoarea corespunzătoare timpului de execuție pentru cazul cel mai defavorabil (WCET: Worst Case Execution Time) în modul de rulare Lo, sistemul va trece în modul de criticalitate Hi, iar task-urile vor fi planificate conform tabelului planificării pentru modul de criticalitate Hi. Pentru fiecare tabel al planificării corespunzător unei unități de procesare, task-urile sunt ordonate crescător în funcție de timpul de start. În continuare, task-ul cu cea mai mică valoare a timpului de start M_i este extras din tabelul planificării, pentru ca prima instanță a acestuia $J_{i,0}$ să fie executată. După finalizarea execuției job-ului $J_{i,0}$, timpul de start al task-ului M_i este recalculat. M_i va fi adăugat din nou în tabelul planificării corespunzător, pe baza expresiei (7-11), urmând ca procesul să se repete (task-ul cu cea mai mică valoare a timpului de start va fi extras din lista sortată de task-uri și executat, ș.a.m.d.).

Algoritmul de mapare se va executa în felul următor:

Fiecare unitate de procesare are asociat un tabel al planificării. Task-urile sunt extrase pe rând din setul de task-uri și adăugate în fiecare tabel. Dacă inițial tabelul planificării nu este gol, se verifică două condiții:

I. Utilizarea pentru unitatea de procesare curentă, care este suma utilizărilor tuturor task-urilor din tabelul planificării, nu trebuie să depășească valoarea 1 [187]:

$$U_{\Gamma_q} \leq 1, \text{ pentru } q = 1, \dots, m \tag{8-1}$$

II. Testul de fezabilitate aplicat subsetului de task-uri de pe unitatea de procesare curentă trebuie să fie pozitiv.

Dacă cele două condiții sunt respectate, task-ul va rămâne în tabelul planificării, iar utilizarea unității de procesare este actualizată. Apoi următorul task este preluat din lista de așteptare și verificat.

În schimb, dacă tabelul planificării este gol, task-ul va fi adăugat în tabel fără a testa cele două condiții, iar utilizarea unității de procesare este actualizată.

Dacă una din cele două condiții nu este respectată, task-ul este extras din tabelul planificării și adăugat în tabelul următoarei unității de procesare, unde se va executa același test.

8.2 Exemple de execuție

Pentru a ilustra metoda de partiționare a task-urilor prezentată în Subcapitolul 8.1, se va considera un set de task-uri planificate pe un sistem cu două unități de procesare, cu niveluri mixte de criticalitate. Tabel 21 conține atât parametrii task-urilor din set, cât și utilizarea pentru fiecare nivel de criticalitate.

Tabel 21. Exemplu de set cu șase task-uri.

Task	T_i	D_i	L_i	$C_{i,L_{Lo}}$	$C_{i,L_{Hi}}$	$U_{i,L_{Lo}}$	$U_{i,L_{Hi}}$
M_1	24	24	Hi	5	6	0.208	0.25
M_2	72	72	Hi	8	9	0.111	0.125
M_3	18	18	Hi	3	4	0.167	0.222
M_4	8	8	Hi	1	2	0.125	0.25
M_5	36	36	Lo	6	-	0.167	-
M_6	12	12	Lo	2	-	0.167	-

Figura 25 ilustrează planificarea task-urilor în modurile de rulare Lo și Hi.

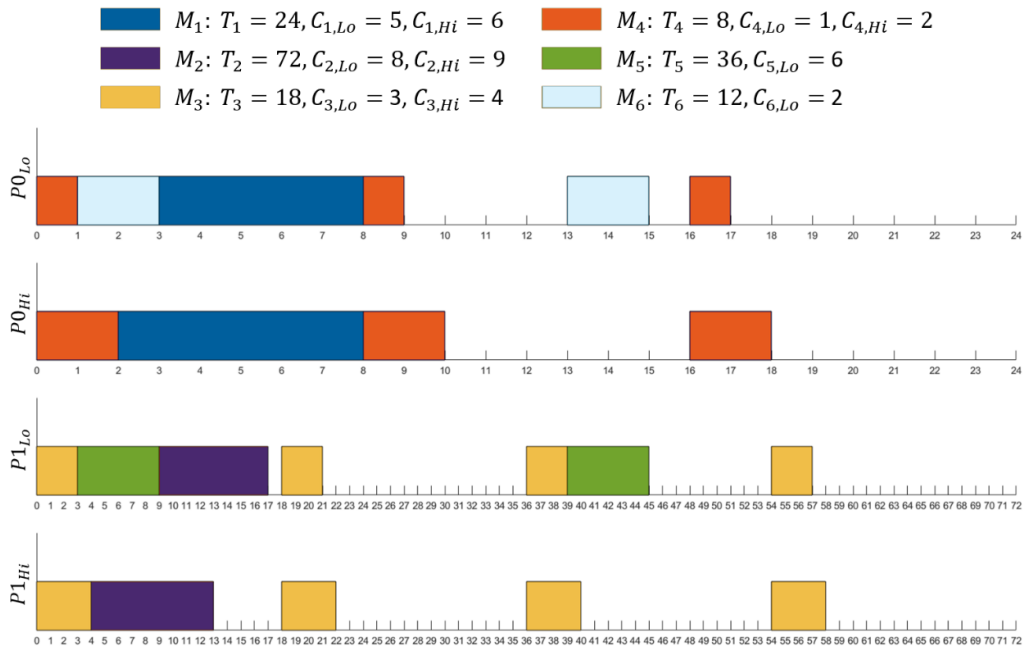


Figura 25. Planificarea exemplului de set cu șase task-uri din Tabel 21 folosind Partitioned Fixed Execution Non-Preemptive Mixed Criticality (P_FENP_MC).

În acest caz, prin aplicarea algoritmului P_FENP_MC se vor obține următoarele rezultate: task-urile M_1, M_4 și M_6 sunt alocate primei unități de procesare (P_0) cu o utilizare totală de 0.5 pentru nivelul de criticalitate Lo și 0.5 pentru nivelul de criticalitate Hi. Task-urile M_2, M_3 și M_5 sunt repartizate pe unitatea de procesare P_1

cu o utilizare totală de 0.445 pentru nivelul de criticalitate Lo și 0.347 pentru nivelul de criticalitate Hi.

De asemenea, pentru condiția I, la calcularea utilizării totale pe fiecare unitate de procesare, se va considera utilizarea totală pentru nivelul de criticalitate Hi în cazul WCET-urilor corespunzătoare nivelului de criticalitate Hi și utilizarea totală pentru nivelul de criticalitate Lo, în cazul WCET-urile corespunzătoare nivelului de criticalitate Lo. Deci, condiția I trebuie respectată atât pentru utilizarea totală în modul de criticalitate Hi, cât și pentru utilizarea totală în modul de criticalitate Lo. Condiția II presupune aplicarea testului de fezabilitate pentru ambele moduri de rulare (Lo și Hi).

8.3 Implementare

În continuare, cele două etape ale algoritmului P_FENP_MC sunt descrise utilizând diagrame. În etapa offline, tabelele planificării pentru fiecare unitate de procesare sunt create folosind funcția de mapare și testul de fezabilitate. Diagrama din Figura 26 ilustrează execuția etapei offline:

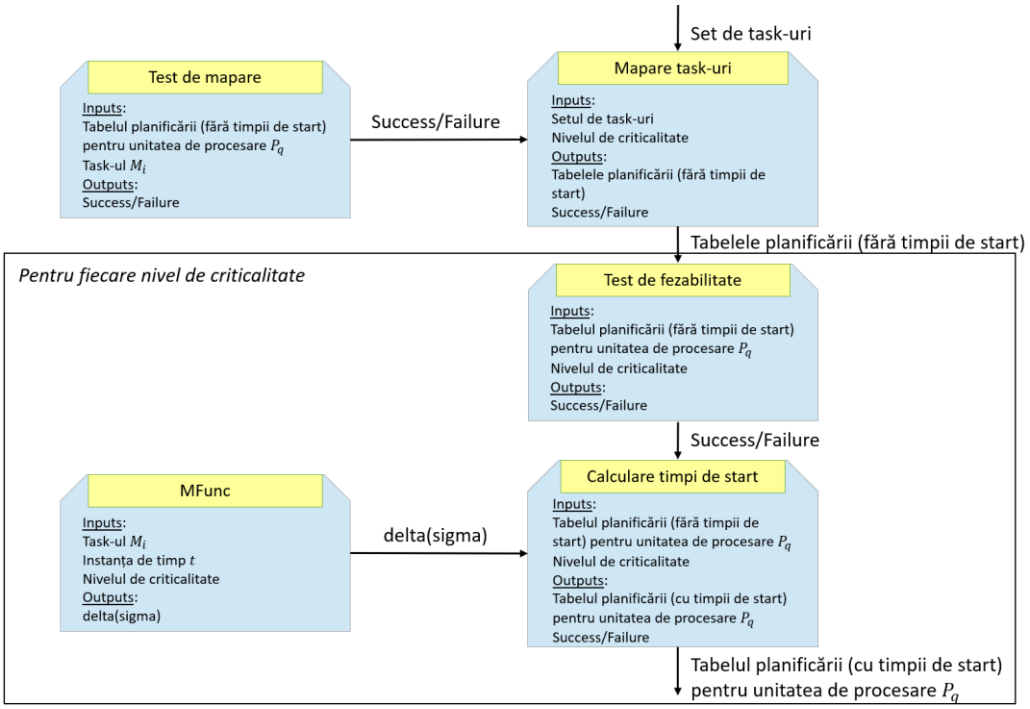


Figura 26. Execuția etapei offline.

Etapa online folosește tabelele planificării create în etapa offline și constă din algoritmul de planificare propriu-zis. Pe fiecare unitate de procesare, tabelul planificării va fi utilizat și actualizat în mod dinamic. În acest tabel, job-urile sunt ordonate crescător în funcție de timpul de start, apoi extrase pe rând pentru a fi executate. După execuția unei instanțe, timpul de start al următorului job este calculat folosind expresia (7-11) și apoi adăugat în tabelul planificării, care va rămâne

sortat în funcție de timpii de start. Figura 27 descrie execuția etapei online a metodei de planificare P_FENP_MC:

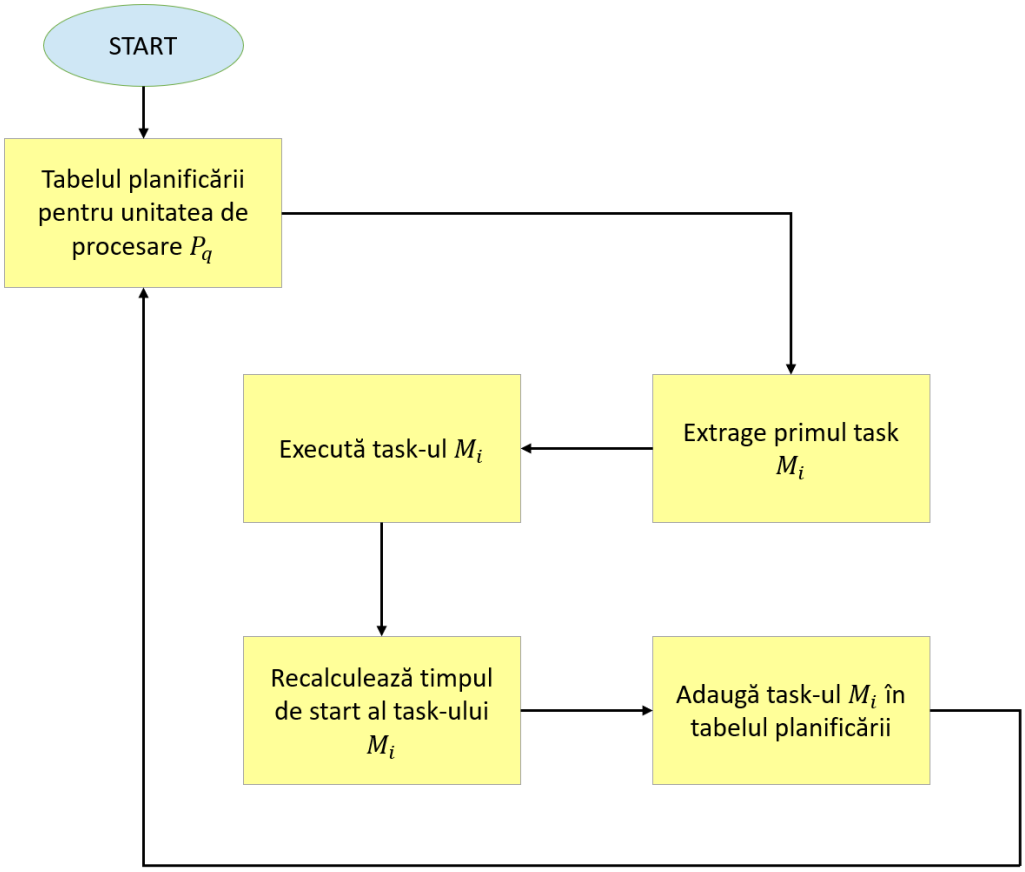


Figura 27. Execuția etapei online.

Testul de fezabilitate efectuat pe fiecare unitate de procesare este descris în Algoritm 9, iar tabelele planificării sunt create de către Algoritm 10.

Algoritm 9. Test de mapare.

Input: Γ_q (tabelul planificării pentru unitatea de procesare q),
M_i (task-ul mapat pe unitatea de procesare q)
Output: $FAILURE$ pentru un test de mapare negativ, $SUCCESS$ altfel

1 $adaugă\ M_i\ în\ \Gamma_q$
2 $sortează\ \Gamma_q\ în\ funcție\ de\ T_i\ în\ ordine\ crescătoare$
3 **for** $i \in \{1, 2, \dots, size\ of\ \Gamma_q - 1\}$ **do**
4 **for** $j \in \{0, 1, \dots, i-1\}$ **do**
5 $gcd \leftarrow cel\ mai\ mare\ divizor\ comun\ al\ lui\ T_i\ şi\ T_j$
6 **if** $C_{i,Lo} + C_{j,Lo} > gcd$ **then**

```
7              elimină M_i din Γ_q
8              return FAILURE
9          end if
10         if L_i = Hi and L_j = Hi and C_i,Hi + C_j,Hi > gcd then
11             return FAILURE
12         end if
13     end for
14 end for
15 return SUCCESS
```

Algoritm 10. P_FENP_MC.

Input: $M \in \{M_0, M_1, \dots, M_{n-1}\}$, unde n este numărul de task-uri
Output: $\Gamma \in \{\Gamma_0, \Gamma_1, \dots, \Gamma_{m-1}\}$, unde m este numărul de unități de procesare
$FAILURE$ dacă nu există suficiente unități de procesare pentru a executa setul de task-uri, $SUCCESS$ altfel

```
1  q ← 0
2  ULo_Γ0 ← 0
3  UHi_Γ0 ← 0
4  for i ∈ {0, 1, ..., n − 1} do
5      for j ∈ {0, 1, ..., q} do
6          tempUtilLo ← C_i,Lo / T_i
7          if L_i = Hi
8              tempUtilHi ← C_i,Hi / T_i
9          else
10             tempUtilHi ← 0
11         end if
12         if ULo_Γj ≤ 1 and UHi_Γj ≤ 1 and mapping_test(Γ_j, M_i) =
   SUCCESS then
13             ULo_Γj ← ULo_Γj + tempUtilLo
14             UHi_Γj ← UHi_Γj + tempUtilHi
15             break
16         end if
17     end for
18     if j > q then
19         if j + 1 > m then
20             return FAILURE
21         end if
22         q ← q + 1
23         ULo_Γq ← tempUtilLo
24         UHi_Γq ← tempUtilHi
25         adaugă M_i în Γ_q
26     end if
27 end for
28 return SUCCESS
```

9 EVALUAREA PERFORMANȚEI

Capitolul curent prezintă rezultatele simulărilor experimentale care au fost efectuate pentru a evalua eficiența algoritmului de planificare Partitioned Fixed Execution Non-Preemptive Mixed Criticality (P_FENP_MC). În acest scop s-a realizat compararea algoritmului, din punctul de vedere al ratei de succes, cu o metodă de planificare pentru MCSs cunoscută și utilizată frecvent în literatura de specialitate, P-EDF-VD (Partitioned Earliest Deadline First with Virtual Deadlines) [97], dar și cu o extensie dezvoltată pentru task-uri periodice pe baza algoritmului table-driven introdus în [28], P-TT-OCBP (Partitioned Time-Triggered Own Criticality Based Priority). Maparea task-urilor pe unitățile de procesare pentru cele două metode s-a realizat utilizând euristica FFD (First Fit Decreasing) [198], cu sortare în funcție de perioadă. În plus, pentru a scoate în evidență execuția fără jitter a algoritmului P_FENP_MC, s-a efectuat compararea sa cu două tehnici de planificare time-triggered, TT-Merge (Time-triggered Merge) și Energy-efficient TT-Merge (Energy-efficient Time-triggered Merge) [105], dar și cu metodele event-driven și table-driven descrise anterior. Aceste simulări au fost executate într-un context non-preemptiv.

Algoritmul pentru platforme cu mai multe unități de procesare P_FENP_MC a fost integrat cu succes în mediul de simulare prezentat în Capitolul 5 pentru sisteme eterogene distribuite cu niveluri mixte de criticalitate (Figura 9).

În cazul simulărilor experimentale din acest capitol, s-a considerat un sistem omogen cu niveluri mixte de criticalitate.

9.1 P-TT-OCBP: Partitioned Time-Triggered Own Criticality Based Priority

Tehnica P-TT-OCBP a fost dezvoltată pe baza algoritmului table-driven introdus în [28]. Astfel, task-urile periodice de pe fiecare unitate de procesare sunt reprezentate ca un set de job-uri independente, obținute prin enumerarea explicită a job-urilor fiecărui task și planificate într-un interval de timp predefinit. Lista de priorități este construită offline, utilizând metoda OCBP (Own Criticality Based Priority) [53], prezentată sub formă de pseudocod în Algoritm 11:

Algoritm 11. OCBP.

Input: Υ_q (lista de job-uri pentru unitatea de procesare q)
Output: Λ_q (lista de priorități pentru unitatea de procesare q)

21 $sortează\ \Upsilon_q\ în\ funcție\ de\ d_{i,k}\ în\ ordine\ crescătoare$
22 **for** $k \in \{0, 1, \dots, size\ of\ \Upsilon_q\}$ **do**
23 $\quad sumLo \leftarrow 0$
24 $\quad sumHi \leftarrow 0$
25 $\quad$ **for** $j \in \{0, 1, \dots, size\ of\ \Upsilon_q\}$ **do**
26 $\quad\quad$ **if** $j \neq k$ **then**
27 $\quad\quad\quad sumLo \leftarrow sumLo + c_{j,Lo}$
28 $\quad\quad\quad$ **if** $L_k = Hi$ **then**
29 $\quad\quad\quad\quad sumHi \leftarrow sumHi + c_{j,Hi}$

```
30            end if
31        end if
32    end for
33    if L_k = Lo and d_k − sumLo ≥ c_{k,Lo} then
34        mută J_k în Λ_q
35    end if
36    if L_k = Hi and d_k − sumLo ≥ c_{k,Lo} and d_k − sumHi ≥ c_{k,Hi} then
37        mută J_k în Λ_q
38    end if
39 end for
```

Algoritmul se repetă până în momentul în care lista de job-uri este goală, în acest caz lista de priorități fiind construită cu succes, sau până când job-urile rămase în listă nu pot fi prioritizate, caz în care setul de job-uri nu este planificabil.

Cele două tabele ale planificării sunt construite pe baza listei de priorități pentru fiecare unitate de procesare. Astfel, primul job este extras din lista de priorități, având timpul de start 0. Pentru următoarele $k+1$ job-uri, timpul de start va depinde de timpul de activare al job-ului respectiv, dar și de momentul la care job-ul precedent își finalizează execuția.

Maparea task-urilor pe unitățile de procesare pentru algoritmul P-TT-OCBP a fost realizată utilizând euristica FFD [198].

9.2 Rata de succes

Analiza experimentală a fost efectuată asupra unor task-uri generate aleatoriu, într-un sistem cu două niveluri de criticalitate $\{Lo, Hi\}$, utilizând algoritmul descris în Subcapitolul 5.2. Astfel, în mod asemănător, parametrii fiecărui task M_i sunt generați după cum urmează:

- Nivelul de criticalitate: $L_i = Hi$ cu o probabilitate de P_{Hi}, altfel $L_i = Lo$.
- Perioada: T_i este o valoare generată aleatoriu uniform în intervalul $[10, 50]$.
- Deadline-ul este egal cu perioada $D_i = T_i$.
- Utilizarea fiecărui task U_{i,L_j} este un vector de valori, de dimensiune l, unde l reprezintă numărul de niveluri de criticalitate.
- WCET-ul pentru nivelul de criticalitate Lo: $C_{i,L_{Lo}} = U_{i,L_{Lo}} \cdot T_i$.
- WCET-ul pentru nivelul de criticalitate Hi: $C_{i,L_{Hi}} = U_{i,L_{Hi}} \cdot T_i$ dacă $L_i = Hi$. Altfel, $C_{i,L_{Hi}} = C_{i,L_{Lo}}$.
- Timpul de start: $S_{i,L_{Lo}} = S_{i,L_{Hi}} = 0$.

Parametrii utilizați pentru generarea setului de task-uri sunt definiți în legenda figurilor (Figura 28 și Figura 29). Fiecare punct de date de pe grafic a fost determinat prin generarea a 100 de seturi de task-uri. În Figura 28 limita superioară a utilizării totale pentru fiecare set de task-uri (axa x) variază de la 0.2 la 0.8 cu un pas de 0.1, valoare înmulțită apoi cu numărul de unități de procesare din sistem și împărțită la 2. Pentru Figura 29 numărul de unități de procesare (axa x) variază de la 2 la 12 cu un pas de 2. Numărul de task-uri din set variază în funcție de limita superioară a utilizării totale. Deci, o valoare mai mare pe axa x, va rezulta într-un număr mai mare de task-uri în set, iar o valoare mai mică, într-un număr mai mic de task-uri.

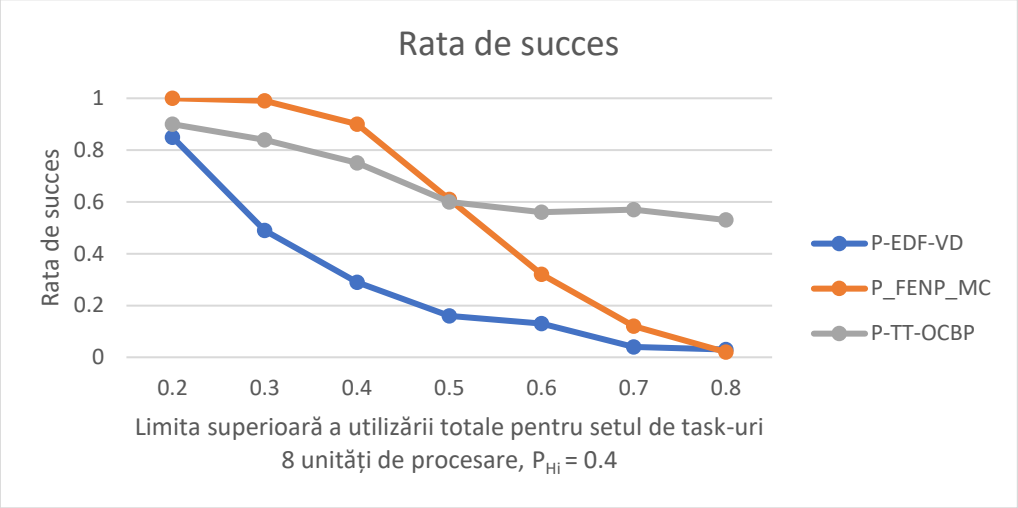


Figura 28. Rata de succes, obținută prin variația limitei superioare a utilizării totale pentru fiecare set de task-uri.

$$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 4.}$$

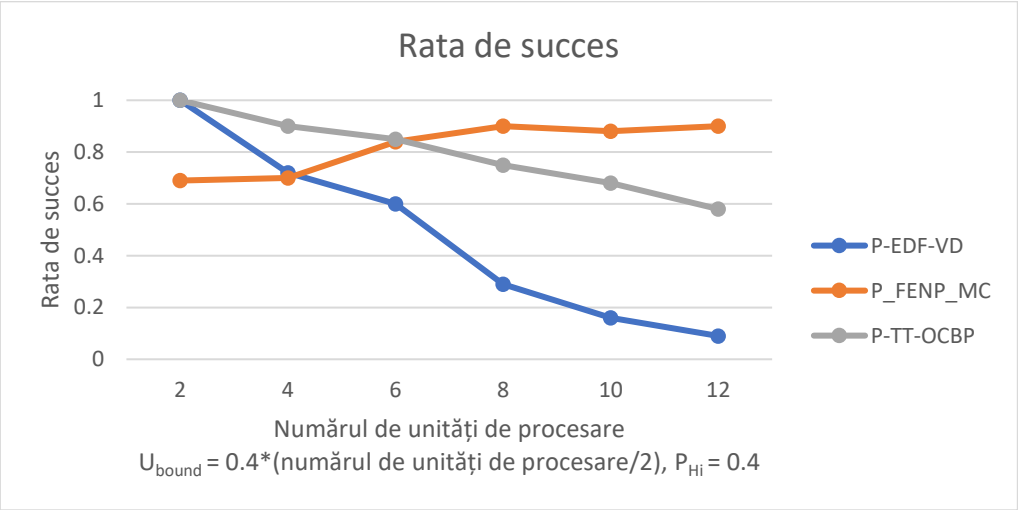


Figura 29. Rata de succes, obținută prin variația numărul de elemente de procesare din sistem.

$$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 4.}$$

În cazul algoritmului P_FENP_MC, cu cât numărul de unități de procesare este mai mare (Figura 29), task-urile sunt mai bine planificate din punctul de vedere al ratei de succes. Având mai multe resurse disponibile, există o șansă mai mare ca fiecare task să fie partiționat pe o unitate de procesare adecvată, în ceea ce privește condițiile I și II (Subcapitolul 8.1). Euristica FFD nu rulează un test de mapare la

partiționarea fiecărui task. În consecință, dacă un număr mare de task-uri este alocat pe o unitate de procesare, algoritmul de planificare local poate să returneze un test de planificare negativ.

Compromisul pentru obținerea unei planificări perfect periodice pe o unitate de procesare este o rată de succes mai mică în comparație cu alte metode. Un algoritm precum EDF-VD poate să ajungă la o rată de succes de 75% în cazul unei utilizări totale de 1, pentru cel mai scăzut nivel de criticalitate [63]. Rezultate comparative sunt mai greu de obținut cu un algoritm time-triggered fără a folosi metode de îmbunătățire a resurselor, cum ar fi de exemplu scalarea frecvenței [28, 105].

Pentru o platformă cu mai multe unități de procesare, rata de succes este influențată atât de algoritmul de planificare local, cât și de euristica utilizată pentru maparea task-urilor pe unitățile de procesare. Dacă se folosește o funcție de partiționare adecvată, se pot obține rezultate comparative din punctul de vedere al ratei de succes, între metode de planificare time-triggered și event-driven. După cum se poate observa în Figura 29, o dată cu creșterea numărului de unități de procesare, algoritmul de planificare propus a obținut rezultate mai bune, prin utilizarea unei funcții de partiționare eficiente, în comparație cu metodele P-EDF-VD și P-TT-OCBP (care aplică euristica de mapare FFD).

9.3 Execuție fără jitter – Test Case

Execuția fără jitter a task-urilor realizată prin proiectarea unei planificări perfect periodice pentru toate nivelurile de criticalitate într-un sistem cu niveluri mixte de criticalitate aduce avantaje în aplicații ce țin de sincronizarea mesajelor, procesarea semnalelor, aplicații de control, sau pur și simplu diferite tipuri de certificări. Astfel, printr-o execuție fără jitter și respectând constrângerile de timp impuse de MCSs, se poate obține un determinism și o predictibilitate deplină în ceea ce privește execuția task-urilor.

În continuare este oferit un exemplu pentru a ilustra execuția fără jitter a unui set de trei task-uri planificat folosind P_FENP_MC și pentru a compara metoda cu alți algoritmi de planificare, într-un sistem cu o singură unitate de procesare, cu niveluri mixte de criticalitate. Tabel 22 conține parametrii de timp ai task-urilor:

Tabel 22. Exemplu de set de task-uri.

Task	T_i	D_i	L_i	$C_{i,L_{Lo}}$	$C_{i,L_{Hi}}$
M_1	8	8	Hi	2	5
M_2	12	12	Lo	1	-
M_3	16	16	Lo	2	-

Planificarea, atât pentru modul de criticalitate Hi, cât și pentru modul de criticalitate Lo, este ilustrată în Figura 30:

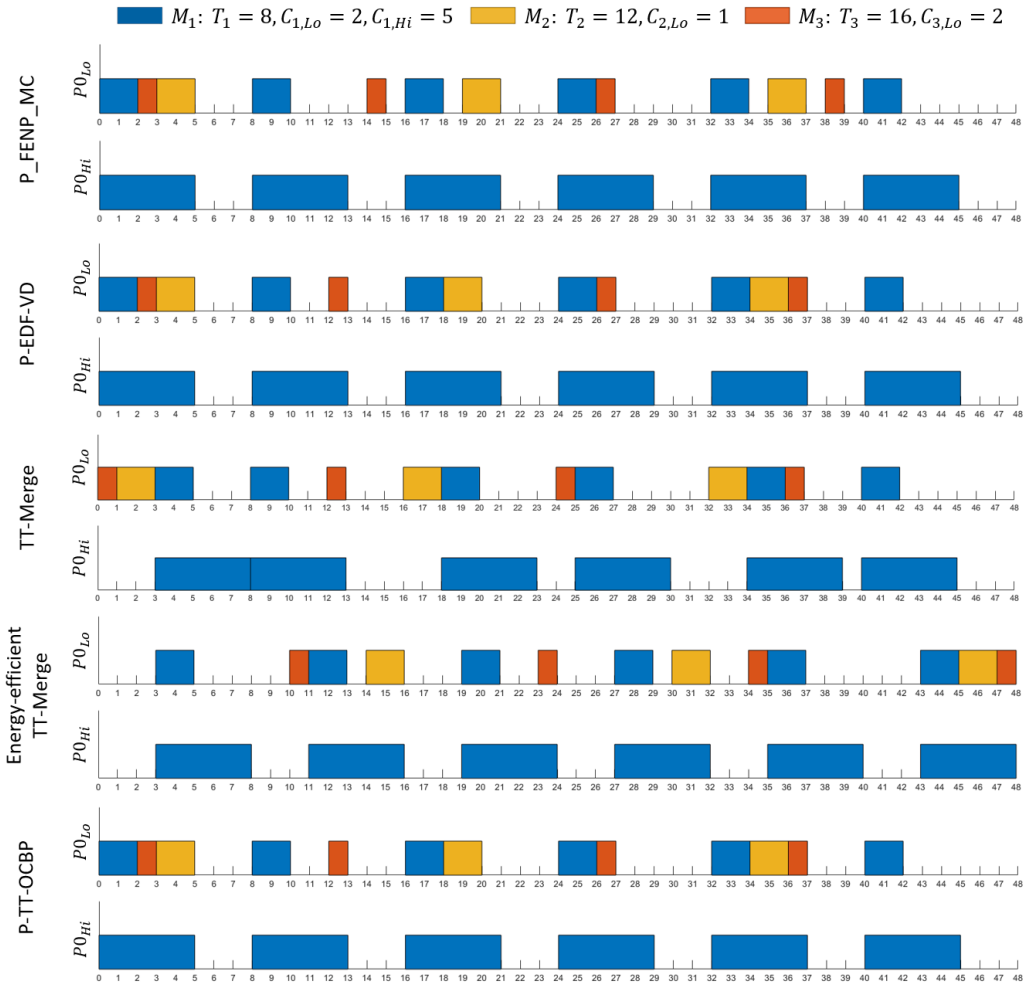


Figura 30. Planificarea exemplului de set de task-uri din Tabel 22 folosind cinci metode: P_FENP_MC, P-EDF-VD (varianta non-preemptivă), TT-Merge, Energy-efficient TT-Merge și P-TT-OCBP.

Tabel 23 conține valorile jitter-ului, obținute aplicând expresia (2-3), pentru exemplul de set de task-uri planificat utilizând P_FENP_MC, P-EDF-VD (varianta non-preemptivă), TT-Merge, Energy-efficient TT-Merge și P-TT-OCBP.

Tabel 23. Valorile jitter-ului pentru exemplul de set de task-uri din Tabel 22 planificat utilizând cinci algoritmi: P_FENP_MC, P-EDF-VD, TT-Merge, Energy-efficient TT-Merge și P-TT-OCBP.

Mod de criticalitate	**Task**	**Valoare jitter**				
		P_FENP_MC	**P-EDF-VD**	**TT-Merge**	**Energy-efficient TT-Merge**	**P-TT-OCBP**
Lo	M_1	0	0	5	0	0
	M_2	0	1	1	1	1
	M_3	0	4	0	2	4
Hi	M_1	0	0	5	0	0

După cum se poate observa din tabelul de mai sus, patru algoritmi (P_FENP_MC, P-EDF-VD, Energy-efficient TT-Merge și P-TT-OCBP) au prezentat o execuție fără jitter a primului task, însă doar P_FENP_MC poate să ofere o execuție fără jitter a tuturor task-urilor din sistem.

10 CONCLUZII ȘI CONTRIBUȚII PERSONALE

10.1 Concluzii

Prima secțiune a tezei abordează integrarea conceptului de MCS în domeniile sistemelor cyber physical, domenii aflate într-o continuă dezvoltare, dar care prezintă probleme de interes. Astfel, am scos în evidență provocările și avantajele integrării MCSs în CPSs. Este de așteptat ca MCSs să joace un rol important în implementări CPSs viitoare, cu capabilități care vor depăși nivelurile actuale de fiabilitate, securitate și funcționalitate. Această abordare are un potențial ridicat deoarece permite integrarea mai multor funcționalități pe aceeași platformă, deci componente diferite pot fi interconectate pentru a rezulta sisteme sigure și cu un grad mai mare de adaptabilitate.

De asemenea, este realizată o prezentare generală a principalelor modele de task-uri existente în literatură pentru MCSs, precum și evoluția lor în timp, dar și o clasificare a algoritmilor de planificare, atât în cazul sistemelor cu o singură unitate de procesare, cât și în cazul sistemelor cu mai multe unități de procesare.

Chiar dacă nu există nici un algoritm de planificare potrivit pentru toate domeniile CPSs, unele metode sunt mai adecvate dacă se iau în considerare doar anumite atribute.

În consecință, un subiect de mare interes este tranziția de la abordările orientate pe aplicație spre tehnici mai generale, standardizate. Standardizarea aplicațiilor și a metodelor de planificare va ușura dezvoltarea diferitelor sisteme de operare, a unor aplicații distribuite care să interconecteze domenii vaste precum IoT, automotive, domeniul medical sau cel militar în sisteme unice, complexe, cu aplicații variate: monitorizarea în timp real, recunoașterea de imagini, controlul de la distanță a unor componente ale sistemului (de exemplu roboți), actuatoare, etc. Prima secțiune a tezei cuprinde sarcinile atribuite obiectivului [O1] al cercetării de doctorat. Această activitate a fost publicată și în [A2]:

[T1.1]. Compararea diferitelor modele de task-uri existente.

Această etapă a fost îndeplinită prin:

- Prezentarea evoluției modelelor de task-uri pentru sisteme cu niveluri mixte de criticalitate din literatura de specialitate.

[T1.2]. Clasificarea algoritmilor de planificare principali în funcție de arhitectura platformei pe care rulează.

În scopul finalizării acestei sarcini:

- S-a realizat clasificarea algoritmilor de planificare utilizați în MCSs, atât pentru sisteme cu o singură unitate de procesare, cât și pentru sisteme cu mai multe unități de procesare.
- Au fost evidențiate principalele tipuri de sisteme cyber physical, precum și o serie de atribute comune care pot influența alegerea algoritmului de planificare într-o astfel de platformă.
- A avut loc o analiză, pe baza acestor atribute, a algoritmilor de planificare clasificați anterior.
- Au fost subliniate o serie de provocări, dar și avantaje ale integrării modelului cu niveluri mixte de criticalitate în CPSs.

A doua secțiune a tezei descrie domeniul sistemelor IoT. O dată cu îmbunătățirile continue aduse tehnologiei IoT, aplicații din ce în ce mai complexe (unele cu cerințe stricte de timp) vor fi dezvoltate pe viitor. În acest scop,

implementarea unor arhitecturi noi IoT de timp real, eficiente, poate oferi un suport tehnologic semnificativ. Combinarea mecanismelor de bază din sistemele cu niveluri mixte de criticalitate și Internet of Things oferă o oportunitate importantă în cercetarea și dezvoltarea aplicațiilor distribuite complexe.

Această teză introduce o formalizare matematică a conceptului de MC-IoT și propune o metodologie eficientă de planificare a aplicațiilor cu niveluri mixte de criticalitate în cadrul nodurilor IoT. Astfel, a doua secțiune sintetizează obiectivul [O2] al tezei de doctorat, care a fost atins prin îndeplinirea sarcinilor propuse, și anume:

[T2.1]. Definirea unui model de task-uri pentru sisteme de timp real distribuite cu niveluri mixte de criticalitate.

Această sarcină a implicat:

- Pe baza modelului clasic de task-uri din MCSs, a fost definit un nou model de task-uri pentru sisteme distribuite, precum și o formalizare matematică a problemei.
- A fost propusă o arhitectură MC-IoT în care task-uri de timp real de criticalitate diferită rulează în cadrul nivelului Edge al unei arhitecturi IoT.
- S-a introdus un nou parametru: scorul de afinitate. Acesta este definit ca un vector de valori, câte o valoare pentru fiecare element de procesare, reprezentând afinitatea task-ului pentru elementul de procesare respectiv.

[T2.2]. Implementarea modelului de task-uri pe un simulator.

Pentru realizarea acestei etape:

- Două metode de setare a scorului de afinitate au fost introduse, și anume: setarea scorului de afinitate pe baza timpului de execuție și setarea scorului de afinitate pe baza nivelului de criticalitate.
- A fost definită o funcție de mapare adecvată, astfel încât să fie respectate cerințele aplicației și constrângerile ce țin de resurse, denumită Best Affinity Fit.
- Cele două metode de setare a scorului de afinitate au fost incorporate în cadrul unui generator de task-uri.
- Metoda de mapare a task-urilor a fost implementată cu ajutorul unui simulator dezvoltat pentru sisteme cu niveluri mixte de criticalitate distribuite.

[T2.3]. Testarea modelului de task-uri propus anterior cu ajutorul mediului de simulare.

În acest scop au fost comparate următoarele metode de mapare:

- Best Affinity Fit (BAF) cu setarea scorului de afinitate pe baza timpului de execuție, și Best Fit Decreasing Utilization (BFDU).
- Best Affinity Fit (BAF) cu setarea scorului de afinitate pe baza nivelului de criticalitate, și Best Fit Decreasing Criticality (BFDC).

Pe măsură ce complexitatea și numărul de aplicații critice de timp real crește, trebuie acordată o atenție deosebită dezvoltării unor tehnici de planificare eficiente și fezabile, în special pentru sistemele critice care rulează într-un mediu controlat de timp. Astfel, în ultima secțiune a tezei a fost propusă o metodă de planificare în timp real, table-driven, non-preemptivă, care garantează o execuție perfect periodică (fără jitter) a task-urilor într-un sistem cu niveluri mixte de criticalitate, atât pentru platforme cu o singură unitate de procesare, cât și pentru platforme cu mai multe unități de procesare. Această secțiune concretizează obiectivele [O3] și [O4], care au fost publicate și în [A3].

Obiectivul [O3] a presupus:

[T3.1]. Realizarea unui algoritm de planificare pentru sisteme de timp real cu niveluri mixte de criticalitate.

Pentru a îndeplini această etapă:

- A fost definit un model de task-uri perfect periodice pentru sisteme de timp real cu niveluri mixte de criticalitate.
- Pe baza metodei de planificare FENP pentru sisteme de timp real cu o singură unitate de procesare care garantează execuția fără jitter a tuturor task-urilor din cadrul platformei, a fost propus un algoritm table-driven, non-preemptiv, pentru sisteme de timp real cu niveluri mixte de criticalitate, FENP_MC.
- Algoritmul FENP_MC a fost extins pentru planificarea în sisteme omogene cu mai multe unități de procesare, cu niveluri mixte de criticalitate (P_FENP_MC), prin definirea unei funcții de mapare.
- Metoda P_FENP_MC a fost implementată în cadrul unui simulator dezvoltat pentru sisteme cu niveluri mixte de criticalitate.

[T3.2]. Conceperea unui test de planificare pentru algoritm.

Această sarcină a fost finaliză prin:

- A fost definit un test de fezabilitate pentru sisteme cu o singură unitate de procesare, cu niveluri mixte de criticalitate.
- S-a introdus un test de mapare pentru sisteme cu mai multe unități de procesare, cu niveluri mixte de criticalitate.
- Cele două teste de planificare au fost incorporate în cadrul simulatorului din etapa anterioară [T3.1].

Obiectivul [O4] a fost îndeplinit prin:

[T4.1]. Testarea, validarea și compararea diferitelor versiuni pentru algoritmul definit la pasul anterior cu ajutorul unui simulator.

În acest scop au fost comparate următoarele metode de planificare:

- Din punctul de vedere al ratei de succes: Partitioned Fixed Execution Non-Preemptive Mixed Criticality (P_FENP_MC), Partitioned Earliest Deadline First with Virtual Deadlines (P-EDF-VD) și Partitioned Time-Triggered Own Criticality Based Priority (P-TT-OCBP).
- Din punctul de vedere al valorii jitter-ului: Partitioned Fixed Execution Non-Preemptive Mixed Criticality (P_FENP_MC), Partitioned Earliest Deadline First with Virtual Deadlines (P-EDF-VD), Time-triggered Merge (TT-Merge), Energy-efficient Time-triggered Merge (Energy-efficient TT-Merge) și Partitioned Time-Triggered Own Criticality Based Priority (P-TT-OCBP).

10.2 Sinteza contribuțiilor

Un scurt rezumat, la nivel de capitole, al contribuțiilor aduse prin această teză este:

- În Capitolul 2:
  - sunt comparate diferitele modele de task-uri existente în domeniul sistemelor cu niveluri mixte de criticalitate ([T1.1]).
  - este realizată o clasificare a algoritmilor de planificare din literatura de specialitate ([T1.2]).

- În Capitolul 3:
  - algoritmii de planificare prezentați în Capitolul 2 sunt comparați pe baza unor caracteristici comune ale sistemelor cyber physical, indiferent de domeniul de aplicabilitate ([T1.2]).
  - sunt scoase în evidență provocările și avantajele integrării conceptului de niveluri mixte de criticalitate în sistemele cyber physical ([T1.2]).
  - sunt analizate principalele caracteristici ale arhitecturilor bazate pe Fog, apoi sunt furnizate exemple de astfel de arhitecturi în IoT ([T2.1]).
  - este prezentată arhitectura IoT cu niveluri mixte de criticalitate (MC-IoT) propusă ([T2.1]).
  - este realizată o formalizare matematică a problemei și sunt introduse ipotezele de lucru ([T2.1]).
- În Capitolul 4:
  - este definit un nou model de task-uri pentru sisteme distribuite cu niveluri mixte de criticalitate, prin introducerea unui nou parametru: scorul de afinitate ([T2.1]).
  - sunt introduse două metode de setare a scorului de afinitate: setarea scorului de afinitate pe baza timpului de execuție și setarea scorului de afinitate pe baza nivelului de criticalitate ([T2.2]).
  - este definită o metodologie de mapare a task-urilor pentru sisteme distribuite cu niveluri mixte de criticalitate, denumită Best Affinity Fit (BAF) ([T2.2]).
- În Capitolul 5:
  - este prezentat mediul de simulare și evaluare a performanțelor, în contextul unui sistem cu niveluri mixte de criticalitate, pentru testarea și validarea diferitelor modele de task-uri și metode de planificare introduse ([T2.2], [T4.1]). Acest mediu de simulare a fost îmbunătățit pentru a putea fi utilizat în cazul sistemelor eterogene distribuite cu niveluri mixte de criticalitate.
- În Capitolul 6:
  - cele două metode de setare a scorului de afinitate sunt integrate în cadrul generatorului de task-uri descris în Capitolul 5 ([T2.2]).
  - metoda de mapare a task-urilor este implementată cu ajutorul mediului de simulare din Capitolul 5 ([T2.2]).
  - este realizată o comparație între metoda de mapare a task-urilor propusă și două euristici de mapare cunoscute ([T2.3]).
- În Capitolul 7:
  - este definit un model de task-uri perfect periodice pentru sisteme de timp real cu niveluri mixte de criticalitate ([T3.1]).
  - este introdus un algoritm table-driven, non-preemptiv pentru sisteme cu o singură unitate de procesare, cu niveluri mixte de criticalitate FENP_MC ([T3.1]).
  - un test de fezabilitate este propus pentru această metodă de planificare ([T3.2]).
- În Capitolul 8:
  - metoda FENP_MC este adaptată pentru sisteme cu mai multe unități de procesare P_FENP_MC prin introducerea unei euristici de mapare ([T3.1]).
  - un test de mapare este definit pentru algoritmul P_FENP_MC ([T3.2]).
- În Capitolul 9:

- cei doi algoritmi FENP_MC, pentru sisteme cu o singură unitate de procesare, și varianta sa partiționată, P_FENP_MC au fost implementați în cadrul simulatorului prezentat în Capitolul 5 ([T4.1]).
- este realizată o comparație din punctul de vedere al ([T4.1]):
  - ratei de succes între metoda de planificare propusă și un algoritm de planificare cunoscut.
  - valorii jitter-ului între metoda de planificare propusă și trei algoritmi de planificare din literatura de specialitate.

10.3 Perspective de dezvoltare

Teza de față propune un model de task-uri care să ia în considerare particularitățile hardware ale elementului de procesare pe care rulează fiecare task, dar și un algoritm de planificare table-driven, non-preemptiv pentru sisteme de timp real cu mai multe unități de procesare, cu niveluri mixte de criticalitate, care oferă o execuție perfect periodică a task-urilor din cadrul platformei.

Ca direcție de dezvoltare se urmărește implementarea practică a algoritmului de planificare, pe baza modelului de task-uri introdus în Subcapitolul 4.1, utilizând LITMUS-RT (Linux Testbed for Multiprocessor Scheduling in Real-Time Systems) [134].

Pentru a facilita procesul de descriere a politicii de planificare, în această teză a fost utilizat un sistem cu două niveluri de criticalitate. Însă, este foarte important ca protocoalele de planificare să fie generalizate pentru un număr realistic de niveluri de criticalitate. Din această cauza, pe viitor se urmărește extinderea algoritmului de planificare pentru sisteme cu $k > 2$ niveluri de criticalitate. Atât modelul de task-uri, cât și algoritmul de planificare introdus în teza de față, au capacitatea de a fi extinse în acest sens.

O altă abordare poate să ia în considerare execuția task-urilor de criticalitate scăzută chiar și după trecerea de la un nivel de criticalitate la altul prin implementarea soluțiilor descrise în Subcapitolul 2.1. Două metode care prezintă potențial ridicat sunt: rularea tuturor task-urilor cu reducerea timpilor de execuție pentru task-urile de criticalitate scăzută sau utilizarea de „slack time” pentru rularea task-urilor de criticalitate scăzută.

De asemenea, în această teză, după o trecere la un nivel de criticalitate ridicat, revenirea sistemului în modul de rulare inițial nu a fost tratată. Ca perspective de dezvoltare, se poate implementa o metodă care să permită acest lucru în cazul în care anumite condiții sunt îndeplinite.

Adițional, o serie de îmbunătățiri pot extinde aplicabilitatea modelului de task-uri și a algoritmului de planificare în domenii, precum:

- Sisteme cyber physical [199] – utilizează în mare parte platforme distribuite, cu mai multe unități de procesare și niveluri mixte de criticalitate datorită puterii de procesare și a mărimii infrastructurilor, dar și a conceptului de izolare temporală și spațială între diferite task-uri. Astfel, algoritmul de planificare introdus în această teză poate fi modificat pentru a permite adaptabilitate și partajarea eficientă a resurselor între diferite niveluri de criticalitate. De asemenea, fiind vorba de platforme distribuite, modelul de task-uri propus oferă o partiționare eficientă a aplicațiilor la nivel de dispozitiv.
- Sisteme multi-agent [200] – întrucât prezintă o implementare ierarhică, în astfel de platforme este de preferat să existe mai multe clase de

algoritmi de planificare, în funcție de cerințele fiecărei componente. Pentru algoritmul introdus, aceste cerințe pot include: execuția perfect periodică a anumitor task-uri, gestionarea eficientă a energiei electrice, sincronizarea între task-uri, etc.

- Sisteme IoT [201] – abordarea poate fi aplicată în cazul sistemelor industriale de control care trebuie să prezinte un comportament complet determinist a anumitor aplicații critice.

BIBLIOGRAFIE

[1] R. Ernst and M. Di Natale, "Mixed criticality systems—A history of misconceptions?," *IEEE Design & Test,* vol. 33, no. 5, pp. 65-74, 2016.

[2] C. Kamienski, M. Jentsch, M. Eisenhauer, J. Kiljander, E. Ferrera, P. Rosengren, J. Thestrup, E. Souto, W. S. Andrade, and D. Sadok, "Application development for the Internet of Things: A context-aware mixed criticality systems development platform," *Computer Communications,* vol. 104, no. pp. 1-16, 2017.

[3] T. Carpenter, J. Hatcliff, and E. Y. Vasserman, "A reference separation architecture for mixed-criticality medical and iot devices," in *Proceedings of the 1st ACM Workshop on the Internet of Safe Things*, 2017, pp. 14-19.

[4] C. Xia, X. Jin, L. Kong, J. Wang, and P. Zeng, "Transmission scheduling for mixed-critical multi-user multiple-input and multiple-output industrial cyber-physical systems," *International Journal of Distributed Sensor Networks,* vol. 13, no. 12, p. 1550147717748910, 2017.

[5] D. Tămaş–Selicean, P. Pop, and W. Steiner, "Design optimization of TTEthernet-based distributed real-time systems," *Real-time systems,* vol. 51, no. 1, pp. 1-35, 2015.

[6] J. Zhan, X. Zhang, W. Jiang, Y. Ma, and K. Jiang, "Energy optimization of security-sensitive mixed-criticality applications for distributed real-time systems," *Journal of Parallel and Distributed Computing,* vol. 117, no. pp. 115-126, 2018.

[7] P. A. Laplante, *Real-time systems design and analysis* vol. 3: Wiley New York, 2004, ISBN:

[8] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR),* vol. 50, no. 6, p. 82, 2018.

[9] K. Velasquez, D. P. Abreu, M. R. Assis, C. Senna, D. F. Aranha, L. F. Bittencourt, N. Laranjeiro, M. Curado, M. Vieira, and E. Monteiro, "Fog orchestration for the internet of everything: state-of-the-art and research challenges," *Journal of Internet Services and Applications,* vol. 9, no. 1, p. 14, 2018.

[10] C.-Y. Chen, M. Hasan, and S. Mohan, "Securing real-time internet-of-things," *Sensors,* vol. 18, no. 12, p. 4356, 2018.

[11] D. Calvaresi, M. Marinoni, A. Sturm, M. Schumacher, and G. Buttazzo, "The challenge of real-time multi-agent systems for enabling IoT and CPS," in *Proceedings of the international conference on web intelligence*, 2017, pp. 356-364.

[12] C. Moratelli, S. Johann, M. Neves, and F. Hessel, "Embedded virtualization for the design of secure IoT applications," in *2016 International Symposium on Rapid System Prototyping (RSP)*, 2016, pp. 1-5.

[13] Y. Yang, K. Wang, G. Zhang, X. Chen, X. Luo, and M.-T. Zhou, "MEETS: Maximal energy efficient task scheduling in homogeneous fog networks," *IEEE Internet of Things Journal,* vol. 5, no. 5, pp. 4076-4087, 2018.

[14] J. Steinbaeck, A. Tengg, G. Holweg, and N. Druml, "A 3D time-of-flight mixed-criticality system for environment perception," in *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 368-374.

[15] A. C. Dimopoulos, G. Bravos, G. Dimitrakopoulos, M. Nikolaidou, V. Nikolopoulos, and D. Anagnostopoulos, "A multi-core context-aware management architecture for mixed-criticality smart building applications," in

2016 11th System of Systems Engineering Conference (SoSE), 2016, pp. 1-6.

[16] G. Bravos, G. Dimitrakopoulos, D. Anagnostopoulos, M. Nikolaidou, C. Kotronis, E. Politi, A. Amira, and F. Bensaali, "Embedded Intelligence in IoT-Based Mixed-Criticality Connected Healthcare Applications: Requirements, Research Achievements and Challenges," no. 2018.

[17] E. A. Capota, C. S. Stangaciu, M. V. Micea, and D.-I. Curiac, "Towards Mixed Criticality Task Scheduling in Cyber Physical Systems: Challenges and Perspectives," *Journal of Systems and Software,* no. 2019.

[18] P. Gaur and M. P. Tahiliani, "Operating systems for IoT devices: A critical survey," in *2015 IEEE Region 10 Symposium*, 2015, pp. 33-36.

[19] J.-E. Kim, T. Abdelzaher, L. Sha, A. Bar-Noy, R. Hobbs, and W. Dron, "On maximizing quality of information for the internet of things: A real-time scheduling perspective," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016, pp. 202-211.

[20] T. Zhang, T. Gong, C. Gu, H. Ji, S. Han, Q. Deng, and X. S. Hu, "Distributed dynamic packet scheduling for handling disturbances in real-time wireless networks," in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017, pp. 261-272.

[21] Z. Hanzálek, T. Tunys, and P. Šůcha, "An analysis of the non-preemptive mixed-criticality match-up scheduling problem," *Journal of Scheduling,* vol. 19, no. 5, pp. 601-607, 2016.

[22] E. A. Capota, C. S. Stangaciu, M. V. Micea, and D.-I. Curiac, "Towards mixed criticality task scheduling in cyber physical systems: Challenges and perspectives," *Journal of Systems and Software,* vol. 156, no. pp. 204-216, 2019.

[23] M.-V. Micea, C.-S. Stangaciu, V. Stangaciu, and D.-I. Curiac, "Novel hybrid scheduling technique for sensor nodes with mixed criticality tasks," *Sensors,* vol. 17, no. 7, p. 1504, 2017.

[24] C. Stangaciu, M. Micea, and V. Cretu, "An Analysis of a Hard Real-Time Execution Environment Extension for FreeRTOS," *Advances in Electrical and Computer Engineering,* vol. 15, no. 3, pp. 79-87, 2015.

[25] C. S. Stangaciu, M. V. Micea, and V. I. Cretu, "Hard real-time execution environment extension for FreeRTOS," in *2014 IEEE International Symposium on Robotic and Sensors Environments (ROSE) Proceedings*, 2014, pp. 124-129.

[26] A. Novak, P. Sucha, and Z. Hanzalek, "Efficient algorithm for jitter minimization in time-triggered periodic mixed-criticality message scheduling problem," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 23-31.

[27] K. Ramamritham and J. A. Stankovic, "Scheduling algorithms and operating systems support for real-time systems," *Proceedings of the IEEE,* vol. 82, no. 1, pp. 55-67, 1994.

[28] S. Baruah and G. Fohler, "Certification-cognizant time-triggered scheduling of mixed-criticality systems," in *2011 IEEE 32nd Real-Time Systems Symposium*, 2011, pp. 3-12.

[29] F. Sagstetter, S. Andalam, P. Waszecki, M. Lukasiewycz, H. Stähle, S. Chakraborty, and A. Knoll, "Schedule integration framework for time-triggered automotive architectures," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1-6.

[30] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 239-243.
[31] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Computing Surveys (CSUR),* vol. 50, no. 6, pp. 1-37, 2017.
[32] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *2010 10th IEEE international conference on computer and information technology*, 2010, pp. 1864-1871.
[33] T. Fleming, "Extending mixed criticality scheduling," University of York, 2013.
[34] M. Zimmer, D. Broman, C. Shaver, and E. A. Lee, "FlexPRET: A processor platform for mixed-criticality systems," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 101-110.
[35] S. K. Baruah, V. Bonifaci, G. d'Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Mixed-criticality scheduling of sporadic task systems," in *European Symposium on Algorithms*, 2011, pp. 555-566.
[36] J. Lee, K.-M. Phan, X. Gu, J. Lee, A. Easwaran, I. Shin, and I. Lee, "MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors," in *2014 IEEE Real-Time Systems Symposium*, 2014, pp. 41-52.
[37] X. Gu, A. Easwaran, K.-M. Phan, and I. Shin, "Resource efficient isolation mechanisms in mixed-criticality scheduling," in *2015 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 13-24.
[38] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *2011 IEEE 32nd Real-Time Systems Symposium*, 2011, pp. 34-43.
[39] H. Su, D. Zhu, and S. Brandt, "An elastic mixed-criticality task model and early-release EDF scheduling algorithms," *ACM Transactions on Design Automation of Electronic Systems (TODAES),* vol. 22, no. 2, pp. 1-25, 2016.
[40] M. Jan, L. Zaourar, and M. Pitel, "Maximizing the execution rate of low criticality tasks in mixed criticality system," *Proc. WMC, RTSS,* no. pp. 43-48, 2013.
[41] C. Gill, J. Orr, and S. Harris, "Supporting graceful degradation through elasticity in mixed-criticality federated scheduling," in *Proc. 6th Workshop on Mixed Criticality Systems (WMC), RTSS*, 2018, pp. 19-24.
[42] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *Workshop on Mixed-Criticality Systems (colocated with RTSS)*, 2013.
[43] T. Fleming and A. Burns, "Incorporating the notion of importance into mixed criticality systems," in *Proc. 2nd Workshop on Mixed Criticality Systems (WMC), RTSS*, 2014, pp. 33-38.
[44] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele, "Interference constraint graph—A new specification for mixed-criticality systems," in *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, 2013, pp. 1-8.
[45] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *2015 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 259-268.
[46] J. Lee, H. S. Chwa, L. T. Phan, I. Shin, and I. Lee, "MC-ADAPT: Adaptive task dropping in mixed-criticality scheduling," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 16, no. 5s, pp. 1-21, 2017.

[47] E. Yip, M. M. Kuo, P. S. Roop, and D. Broman, "Relaxing the synchronous approach for mixed-criticality systems," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 89-100.

[48] D. de Niz and L. T. Phan, "Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 111-122.

[49] H. Xu and A. Burns, "Semi-partitioned model for dual-core mixed criticality system," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, 2015, pp. 257-266.

[50] J. Lee and K. G. Shin, "Development and use of a new task model for cyber-physical systems: A real-time scheduling perspective," *Journal of Systems and Software,* vol. 126, no. pp. 45-56, 2017.

[51] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *IEEE real-time systems symposium*, 1991, pp. 129-139.

[52] A. K.-L. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Massachusetts Institute of Technology, 1983.

[53] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, pp. 13-22.

[54] L. Zeng, C. Xu, and R. Li, "Partition and Scheduling of the Mixed-Criticality Tasks Based on Probability," *IEEE Access,* vol. 7, no. pp. 87837-87848, 2019.

[55] A. Burns, "An augmented model for mixed criticality," in *Mixed Criticality on Multicore/Manycore Platforms (Dagstuhl Seminar 15121)*, 2015, pp. 92-93.

[56] Z. Li and S. He, "Fixed-priority scheduling for two-phase mixed-criticality systems," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 17, no. 2, p. 35, 2018.

[57] H. Su and D. Zhu, "An elastic mixed-criticality task model and its scheduling algorithm," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 147-152.

[58] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Transactions on Computers,* vol. 44, no. 12, pp. 1443-1451, 1995.

[59] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011, pp. 71-80.

[60] P. Ekberg and W. Yi, "Schedulability analysis of a graph-based task model for mixed-criticality systems," *Real-time systems,* vol. 52, no. 1, pp. 1-37, 2016.

[61] S. Asyaban and M. Kargahi, "An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems," *Real-time systems,* vol. 54, no. 1, pp. 32-90, 2018.

[62] A. Crespo, A. Alonso, M. Marcos, A. Juan, and P. Balbastre, "Mixed criticality in control systems," *IFAC Proceedings Volumes,* vol. 47, no. 3, pp. 12261-12271, 2014.

[63] S. Baruah, V. Bonifaci, G. DAngelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *2012 24th Euromicro Conference on Real-Time Systems*, 2012, pp. 145-154.

[64] P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-time systems,* vol. 50, no. 1, pp. 48-86, 2014.
[65] P. Rodriguez, L. George, Y. Abdeddaïm, and J. Goossens, "Multicriteria evaluation of partitioned edf-vd for mixed-criticality systems upon identical processors," in *Workshop on Mixed Criticality Systems*, 2013.
[66] C. C. Arlock and E. Linderoth-Olson, "A Practical Comparison of Scheduling Algorithms for Mixed Criticality Embedded Systems," no. 2014.
[67] K. Lakshmanan, D. de Niz, and R. Rajkumar, "Mixed-criticality task synchronization in zero-slack scheduling," in *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2011, pp. 47-56.
[68] F. Santy, L. George, P. Thierry, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with fp," in *2012 24th Euromicro Conference on Real-Time Systems*, 2012, pp. 155-165.
[69] S. K. Baruah and Z. Guo, "Mixed-criticality job models: a comparison," 2015.
[70] M. Völp, M. Hähnel, and A. Lackorzynski, "Has energy surpassed timeliness? Scheduling energy-constrained mixed-criticality systems," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 275-284.
[71] I. Ali, J.-h. Seo, and K. H. Kim, "A dynamic power-aware scheduling of mixed-criticality real-time systems," in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, 2015, pp. 438-445.
[72] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele, "Energy efficient dvfs scheduling for mixed-criticality systems," in *Proceedings of the 14th International Conference on Embedded Software*, 2014, p. 11.
[73] J. Lee, H. S. Chwa, L. T. Phan, I. Shin, and I. Lee, "MC-ADAPT: Adaptive task dropping in mixed-criticality scheduling," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 16, no. 5s, p. 163, 2017.
[74] T. Park and S. Kim, "Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems," in *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, 2011, pp. 253-262.
[75] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems," in *2011 IEEE 32nd Real-Time Systems Symposium*, 2011, pp. 13-23.
[76] E. A. Capota, C. S. Stangaciu, M. V. Micea, and V. I. Cretu, "P_FENP: A Multiprocessor Real-Time Scheduling Algorithm," in *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, 2018, pp. 000509-000514.
[77] S. Baruah, B. Chattopadhyay, H. Li, and I. Shin, "Mixed-criticality scheduling on multiprocessors," *Real-time systems,* vol. 50, no. 1, pp. 142-177, 2014.
[78] C. Lu, "Mixed-criticality scheduling of an autonomous driving car," *Master's Thesis in Institute for Integrated Systems. Technische Universität München, Department of Electrical and Computer Engineering,* no. 2016.
[79] G. Giannopoulou, "Implementation of Mixed-Criticality Applications on Multi-Core Architectures," ETH Zurich, 2017.
[80] P. Huang, G. Giannopoulou, R. Ahmed, D. B. Bartolini, and L. Thiele, "An isolation scheduling model for multicores," in *2015 IEEE Real-Time Systems Symposium*, 2015, pp. 141-152.

[81] G. Giannopoulou, P. Huang, R. Ahmed, D. B. Bartolini, and L. Thiele, "Isolation scheduling on multicores: model and scheduling approaches," *Real-time systems,* vol. 53, no. 4, pp. 614-667, 2017.
[82] S. K. Baruah, "Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors," *IEEE Transactions on Computers,* vol. 53, no. 6, pp. 781-784, 2004.
[83] S. Funk, G. Levin, C. Sadowski, I. Pye, and S. Brandt, "DP-Fair: a unifying theory for optimal hard real-time multiprocessor scheduling," *Real-time systems,* vol. 47, no. 5, p. 389, 2011.
[84] M. A. Awan, K. Bletsas, P. F. Souto, and E. Tovar, "Semi-partitioned mixed-criticality scheduling," in *International Conference on Architecture of Computing Systems*, 2017, pp. 205-218.
[85] A. Ali and K. H. Kim, "Cluster-based multicore real-time mixed-criticality scheduling," *Journal of Systems Architecture,* vol. 79, no. pp. 45-58, 2017.
[86] E. K. Burke, M. R. Hyde, and G. Kendall, "Evolving bin packing heuristics with genetic programming," in *Parallel Problem Solving from Nature-PPSN IX*, ed: Springer, 2006, ISBN: pp. 860-869.
[87] S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: a load sharing facility for large, heterogeneous distributed computer systems," *Software: practice and Experience,* vol. 23, no. 12, pp. 1305-1336, 1993.
[88] H. Pérez, J. J. Gutiérrez, S. Peiró, and A. Crespo, "Distributed architecture for developing mixed-criticality systems in multi-core platforms," *Journal of Systems and Software,* vol. 123, no. pp. 145-159, 2017.
[89] A. Crespo, I. Ripoll, and M. Masmano, "Partitioned embedded architecture based on hypervisor: The xtratum approach," in *2010 European Dependable Computing Conference*, 2010, pp. 67-72.
[90] O. A. Specification, "Data distribution service for real-time systems version 1.2," *Object Management Group (OMG),* no. 2007.
[91] A. Larrucea, I. Martinez, V. Brocal, S. Peirò, H. Ahmadian, J. Perez, and R. Obermaisser, "DREAMS: Cross-domain mixed-criticality patterns," 2016.
[92] K. Lakshmanan, D. De Niz, R. Rajkumar, and G. Moreno, "Resource allocation in distributed mixed-criticality cyber-physical systems," in *2010 IEEE 30th International Conference on Distributed Computing Systems*, 2010, pp. 169-178.
[93] D. De Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *2009 30th IEEE Real-Time Systems Symposium*, 2009, pp. 291-300.
[94] G. Xie, G. Zeng, L. Liu, R. Li, and K. Li, "High performance real-time scheduling of multiple mixed-criticality functions in heterogeneous distributed embedded systems," *Journal of Systems Architecture,* vol. 70, no. pp. 3-14, 2016.
[95] G. Xie, G. Zeng, Z. Li, R. Li, and K. Li, "Adaptive dynamic scheduling on multifunctional mixed-criticality automotive cyber-physical systems," *IEEE Transactions on Vehicular Technology,* vol. 66, no. 8, pp. 6676-6692, 2017.
[96] J. Ren and L. T. X. Phan, "Mixed-criticality scheduling on multiprocessors using task grouping," in *2015 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 25-34.
[97] J.-J. Han, X. Tao, D. Zhu, H. Aydin, Z. Shao, and L. T. Yang, "Multicore mixed-criticality systems: Partitioned scheduling and utilization bound," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 37, no. 1, pp. 21-34, 2017.

[98] S. Narayana, P. Huang, G. Giannopoulou, L. Thiele, and R. V. Prasad, "Exploring energy saving for mixed-criticality systems on multi-cores," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016, pp. 1-12.
[99] R. Gratia, T. Robert, and L. Pautet, "Scheduling of mixed-criticality systems with RUN," in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, 2015, pp. 1-8.
[100] V. Legout, M. Jan, and L. Pautet, "Mixed-criticality multiprocessor real-time systems: Energy consumption vs deadline misses," in *First Workshop on Real-Time Mixed Criticality Systems (ReTiMiCS)*, 2013, pp. 1-6.
[101] S. Baruah, A. Burns, and Z. Guo, "Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors," in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016, pp. 131-138.
[102] A. Burns and R. I. Davis, "Response time analysis for mixed criticality systems with arbitrary deadlines," in *5th International Workshop on Mixed Criticality Systems (WMC 2017)*, 2017.
[103] J. Theis, G. Fohler, and S. Baruah, "Schedule table generation for time-triggered mixed criticality systems," *Proc. WMC, RTSS,* no. pp. 79-84, 2013.
[104] L. Behera and P. Bhaduri, "Time-triggered scheduling for multiprocessor mixed-criticality systems," in *International Conference on Distributed Computing and Internet Technology*, 2018, pp. 135-151.
[105] L. Behera and P. Bhaduri, "An energy-efficient time-triggered scheduling algorithm for mixed-criticality systems," *Design Automation for Embedded Systems,* no. pp. 1-31, 2019.
[106] S. Baruah, G. Buttazzo, S. Gorinsky, and G. Lipari, "Scheduling periodic task systems to minimize output jitter," in *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306)*, 1999, pp. 62-69.
[107] J. L. Herman, C. J. Kenna, M. S. Mollison, J. H. Anderson, and D. M. Johnson, "RTOS support for multicore mixed-criticality systems," in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, 2012, pp. 197-208.
[108] D. Isovic and G. Fohler, "Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints," in *Proceedings 21st IEEE Real-Time Systems Symposium*, 2000, pp. 207-216.
[109] A. Burns, T. Fleming, and S. Baruah, "Cyclic executives, multi-core platforms and mixed criticality applications," in *2015 27th Euromicro Conference on Real-Time Systems*, 2015, pp. 3-12.
[110] F. Scheler and W. Schröder-Preikschat, "Time-Triggered vs. Event-Triggered: A matter of configuration?," in *ITG FA 6.2 Workshop on Model-Based Testing, GI/ITG Workshop on Non-Functional Properties of Embedded Systems, 13th GI/ITG Conference Measuring, Modelling, and Evaluation of Computer and Communications*, 2006, pp. 1-6.
[111] R. Medina, E. Borde, and L. Pautet, "Generalized Mixed-Criticality Static Scheduling for Periodic Directed Acyclic Graphs on Multi-Core Processors," *IEEE Transactions on Computers,* no. 2020.
[112] J. Fonseca, G. Nelissen, V. Nelis, and L. M. Pinho, "Response time analysis of sporadic dag tasks under partitioned scheduling," in *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, 2016, pp. 1-10.

[113] R. Schneider, D. Goswami, A. Masrur, M. Becker, and S. Chakraborty, "Multi-layered scheduling of mixed-criticality cyber-physical systems," *Journal of Systems Architecture,* vol. 59, no. 10, pp. 1215-1230, 2013.
[114] D. Calvaresi, M. Marinoni, L. Lustrissimini, K. Appoggetti, P. Sernani, A. F. Dragoni, M. Schumacher, and G. Buttazzo, "Local scheduling in multi-agent systems: getting ready for safety-critical scenarios," in *Multi-Agent Systems and Agreement Technologies*, ed: Springer, 2017, ISBN: pp. 96-111.
[115] M. Stigge and W. Yi, "Graph-based models for real-time workload: a survey," *Real-time systems,* vol. 51, no. 5, pp. 602-636, 2015.
[116] T. D. Braun, H. Siegal, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, and D. Hensgen, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems," in *Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)*, 1999, pp. 15-29.
[117] D. Goswami, R. Schneider, A. Masrur, M. Lukasiewycz, S. Chakraborty, H. Voit, and A. Annaswamy, "Challenges in automotive cyber-physical systems design," in *2012 International Conference on Embedded Computer Systems (SAMOS)*, 2012, pp. 346-354.
[118] M. Lukasiewycz, R. Schneider, D. Goswami, and S. Chakraborty, "Modular scheduling of distributed heterogeneous time-triggered automotive systems," in *17th Asia and South Pacific design automation conference*, 2012, pp. 665-670.
[119] J. Kim, H. Kim, K. Lakshmanan, and R. R. Rajkumar, "Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car," in *Proceedings of the ACM/IEEE 4th international conference on cyber-physical systems*, 2013, pp. 31-40.
[120] S. Chakraborty, M. A. Al Faruque, W. Chang, D. Goswami, M. Wolf, and Q. Zhu, "Automotive cyber–physical systems: A tutorial introduction," *IEEE Design & Test,* vol. 33, no. 4, pp. 92-108, 2016.
[121] A. Easwaran and I. Lee, "Compositional schedulability analysis for cyber-physical systems," *ACM Sigbed Review,* vol. 5, no. 1, p. 6, 2008.
[122] A. Easwaran, I. Lee, O. Sokolsky, and S. Vestal, "A compositional scheduling framework for digital avionics systems," in *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2009, pp. 371-380.
[123] Y.-H. Lee, D. Kim, M. Younis, J. Zhou, and J. McElroy, "Resource scheduling in dependable integrated modular avionics," in *Proceeding International Conference on Dependable Systems and Networks. DSN 2000*, 2000, pp. 14-23.
[124] J. C. Pemberton and L. Greenwald, "On the need for dynamic scheduling of imaging satellites," *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences,* vol. 34, no. 1, pp. 165-171, 2002.
[125] A. M. Cheng, "Cyber-physical medical and medication systems," in *2008 The 28th International Conference on Distributed Computing Systems Workshops*, 2008, pp. 529-532.
[126] S. A. Haque, S. M. Aziz, and M. Rahman, "Review of cyber-physical system in healthcare," *International Journal of Distributed Sensor Networks,* vol. 10, no. 4, p. 217415, 2014.
[127] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda, "Cyber-physical systems in manufacturing," *Cirp Annals,* vol. 65, no. 2, pp. 621-641, 2016.

[128] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen, "Real-time wireless sensor-actuator networks for industrial cyber-physical systems," *Proceedings of the IEEE,* vol. 104, no. 5, pp. 1013-1024, 2015.
[129] C. Wu, M. Sha, D. Gunatilaka, A. Saifullah, C. Lu, and Y. Chen, "Analysis of EDF scheduling for wireless sensor-actuator networks," in *2014 IEEE 22nd International Symposium of Quality of Service (IWQoS)*, 2014, pp. 31-40.
[130] J. Shi, M. Sha, and Z. Yang, "Distributed Graph Routing and Scheduling for Industrial Wireless Sensor-Actuator Networks," *IEEE/ACM Transactions on Networking,* no. 2019.
[131] M. Fakih, A. Lenz, M. Azkarate-Askasua, J. Coronel, A. Crespo, S. Davidmann, J. C. D. Garcia, N. G. Romero, K. Grüttner, and S. Schreiner, "SAFEPOWER project: Architecture for safe and power-efficient mixed-criticality systems," *Microprocessors and Microsystems,* vol. 52, no. pp. 89-105, 2017.
[132] A. Taherin, M. Salehi, and A. Ejlali, "Reliability-aware energy management in mixed-criticality systems," *IEEE Transactions on Sustainable Computing,* vol. 3, no. 3, pp. 195-208, 2018.
[133] I. Gerostathopoulos, T. Bures, P. Hnetynka, J. Keznikl, M. Kit, F. Plasil, and N. Plouzeau, "Self-adaptation in software-intensive cyber–physical systems: From system goals to architecture configurations," *Journal of Systems and Software,* vol. 122, no. pp. 378-397, 2016.
[134] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, "Litmus^ rt: A testbed for empirically comparing real-time multiprocessor schedulers," in *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, 2006, pp. 111-126.
[135] A. Cavalcanti and R. A. Freitas, "Nanorobotics control design: A collective behavior approach for medicine," *IEEE Transactions on Nanobioscience,* vol. 4, no. 2, pp. 133-140, 2005.
[136] N. Accettura, M. R. Palattella, G. Boggia, L. A. Grieco, and M. Dohler, "Decentralized traffic aware scheduling for multi-hop low power lossy networks in the internet of things," in *2013 IEEE 14th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, 2013, pp. 1-6.
[137] S. Malik, S. Ahmad, I. Ullah, D. H. Park, and D. Kim, "An Adaptive Emergency First Intelligent Scheduling Algorithm for Efficient Task Management and Scheduling in Hybrid of Hard Real-Time and Soft Real-Time Embedded IoT Systems," *Sustainability,* vol. 11, no. 8, p. 2192, 2019.
[138] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review,* vol. 44, no. 5, pp. 27-32, 2014.
[139] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Transactions on Emerging Topics in Computing,* vol. 5, no. 1, pp. 108-119, 2015.
[140] D. B. Rawat, C. Brecher, H. Song, and S. Jeschke, *Industrial Internet of Things: Cybermanufacturing Systems*: Springer, 2017, ISBN: 3319425587.
[141] R. Squire and H. Song, "Cyber-physical systems opportunities in the chemical industry: A security and emergency management example," *Process Safety Progress,* vol. 33, no. 4, pp. 329-332, 2014.

[142] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future generation computer systems,* vol. 29, no. 7, pp. 1645-1660, 2013.
[143] S. K. Datta, C. Bonnet, and N. Nikaein, "An IoT gateway centric architecture to provide novel M2M services," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 514-519.
[144] S. K. Datta, C. Bonnet, and J. Haerri, "Fog computing architecture to enable consumer centric internet of things services," in *2015 International Symposium on Consumer Electronics (ISCE)*, 2015, pp. 1-2.
[145] P. Hu, S. Dhelim, H. Ning, and T. Qiu, "Survey on fog computing: architecture, key technologies, applications and open issues," *Journal of network and computer applications,* vol. 98, no. pp. 27-42, 2017.
[146] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing,* vol. 4, no. 2, pp. 26-35, 2017.
[147] N. Mohan and J. Kangasharju, "Edge-Fog cloud: A distributed cloud for Internet of Things computations," in *2016 Cloudification of the Internet of Things (CIoT)*, 2016, pp. 1-6.
[148] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE communications surveys & tutorials,* vol. 20, no. 1, pp. 416-464, 2017.
[149] M. Firdhous, O. Ghazali, and S. Hassan, "Fog computing: Will it be the future of cloud computing?," 2014.
[150] P. More, "Review of implementing fog computing," *International Journal of Research in Engineering and Technology,* vol. 4, no. 06, pp. 335-338, 2015.
[151] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for Internet of Things: a primer," *Digital Communications and Networks,* vol. 4, no. 2, pp. 77-86, 2018.
[152] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, 2015, pp. 73-78.
[153] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid—The new and improved power grid: A survey," *IEEE communications surveys & tutorials,* vol. 14, no. 4, pp. 944-980, 2011.
[154] F. Y. Okay and S. Ozdemir, "A fog computing based smart grid model," in *2016 international symposium on networks, computers and communications (ISNCC)*, 2016, pp. 1-6.
[155] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," in *Internet of things*, ed: Elsevier, 2016, ISBN: pp. 61-75.
[156] T. Park and K. G. Shin, "LiSP: A lightweight security protocol for wireless sensor networks," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 3, no. 3, pp. 634-660, 2004.
[157] R. Mahmud, F. L. Koch, and R. Buyya, "Cloud-fog interoperability in IoT-enabled healthcare solutions," in *Proceedings of the 19th international conference on distributed computing and networking*, 2018, pp. 1-10.
[158] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of everything*, ed: Springer, 2018, ISBN: pp. 103-130.

[159] M. García-Valls, T. Cucinotta, and C. Lu, "Challenges in real-time virtualization and predictable cloud computing," *Journal of Systems Architecture,* vol. 60, no. 9, pp. 726-740, 2014.
[160] H. F. Atlam, R. J. Walters, and G. B. Wills, "Fog computing and the internet of things: a review," *big data and cognitive computing,* vol. 2, no. 2, p. 10, 2018.
[161] M. Gupta and R. Sandhu, "Authorization framework for secure cloud assisted connected cars and vehicular internet of things," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, 2018, pp. 193-204.
[162] Y. Xiao and C. Zhu, "Vehicular fog computing: Vision and challenges," in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017, pp. 6-9.
[163] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylae-Jaeaeski, "Fog following me: Latency and quality balanced task allocation in vehicular fog computing," in *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2018, pp. 1-9.
[164] S. S. Adhatarao, M. Arumaithurai, and X. Fu, "FOGG: A fog computing based gateway to integrate sensor networks to Internet," in *2017 29th International Teletraffic Congress (ITC 29)*, 2017, pp. 42-47.
[165] J. Liu, J. Li, L. Zhang, F. Dai, Y. Zhang, X. Meng, and J. Shen, "Secure intelligent traffic light control using fog computing," *Future generation computer systems,* vol. 78, no. pp. 817-824, 2018.
[166] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big data and internet of things: A roadmap for smart environments*, ed: Springer, 2014, ISBN: pp. 169-186.
[167] S. S. I. Samuel, "A review of connectivity challenges in IoT-smart home," in *2016 3rd MEC International conference on big data and smart city (ICBDSC)*, 2016, pp. 1-4.
[168] A. Yassine, S. Singh, M. S. Hossain, and G. Muhammad, "IoT big data analytics for smart homes with fog and cloud computing," *Future generation computer systems,* vol. 91, no. pp. 563-573, 2019.
[169] B. R. Stojkoska and K. Trivodaliev, "Enabling internet of things for smart homes through fog computing," in *2017 25th Telecommunication Forum (TELFOR)*, 2017, pp. 1-4.
[170] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare internet of things: A case study on ecg feature extraction," in *2015 IEEE international conference on computer and information technology; ubiquitous computing and communications; dependable, autonomic and secure computing; pervasive intelligence and computing*, 2015, pp. 356-363.
[171] A. A. Mutlag, M. K. A. Ghani, N. a. Arunkumar, M. A. Mohammed, and O. Mohd, "Enabling technologies for fog computing in healthcare IoT systems," *Future generation computer systems,* vol. 90, no. pp. 62-78, 2019.
[172] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, "Fog computing in healthcare–a review and discussion," *IEEE Access,* vol. 5, no. pp. 9206-9222, 2017.
[173] L. Huang, G. Li, J. Wu, L. Li, J. Li, and R. Morello, "Software-defined QoS provisioning for fog computing advanced wireless sensor networks," in *2016 Ieee Sensors*, 2016, pp. 1-3.

[174] Q. Yaseen, F. AlBalas, Y. Jararweh, and M. Al-Ayyoub, "A fog computing based system for selective forwarding detection in mobile wireless sensor networks," in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, 2016, pp. 256-262.
[175] N. Peter, "Fog computing and its real time applications," *International Journal of Emerging Technology and Advanced Engineering,* vol. 5, no. 6, pp. 266-269, 2015.
[176] P. O'donovan, C. Gallagher, K. Bruton, and D. T. O'Sullivan, "A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications," *Manufacturing Letters,* vol. 15, no. pp. 139-142, 2018.
[177] M. S. de Brito, S. Hoque, R. Steinke, A. Willner, and T. Magedanz, "Application of the fog computing paradigm to smart factories and cyber-physical systems," *Transactions on Emerging Telecommunications Technologies,* vol. 29, no. 4, p. e3184, 2018.
[178] T. M. Fernández-Caramés, P. Fraga-Lamas, M. Suárez-Albela, and M. Vilar-Montesinos, "A fog computing and cloudlet based augmented reality system for the industry 4.0 shipyard," *Sensors,* vol. 18, no. 6, p. 1798, 2018.
[179] J. K. Zao, T. T. Gan, C. K. You, S. J. R. Méndez, C. E. Chung, Y. Te Wang, T. Mullen, and T. P. Jung, "Augmented brain computer interaction based on fog computing and linked data," in *2014 International Conference on Intelligent Environments*, 2014, pp. 374-377.
[180] E. McKenna, I. Richardson, and M. Thomson, "Smart meter data: Balancing consumer privacy concerns with legitimate applications," *Energy Policy,* vol. 41, no. pp. 807-814, 2012.
[181] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, "Fog computing: Focusing on mobile users at the edge," *arXiv preprint arXiv:1502.01815,* no. 2015.
[182] N. B. Truong, G. M. Lee, and Y. Ghamri-Doudane, "Software defined networking-based vehicular adhoc network with fog computing," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 1202-1207.
[183] J. Zhu, D. S. Chan, M. S. Prabhu, P. Natarajan, H. Hu, and F. Bonomi, "Improving web sites performance using edge servers in fog computing architecture," in *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, 2013, pp. 320-323.
[184] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012, pp. 13-16.
[185] Y. N. Krishnan, C. N. Bhagwat, and A. P. Utpat, "Fog computing—Network based cloud computing," in *2015 2nd International Conference on Electronics and Communication Systems (ICECS)*, 2015, pp. 250-251.
[186] O. R. Kelly, H. Aydin, and B. Zhao, "On partitioned scheduling of fixed-priority mixed-criticality task sets," in *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, 2011, pp. 1051-1059.
[187] D. Socci, "Scheduling of certifiable mixed-criticality systems," Grenoble Alpes, 2016.
[188] A. Sabu, B. Raveendran, and R. Ghosh, "SMILEY: a mixed-criticality real-time task scheduler for multicore systems," in *2018 IEEE/ACM 22nd International*

Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2018, pp. 1-5.

[189] H. Li and S. Baruah, "Outstanding paper award: Global mixed-criticality scheduling on multiprocessors," in *2012 24th Euromicro Conference on Real-Time Systems*, 2012, pp. 166-175.

[190] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Improving the scheduling of certifiable mixed-criticality sporadic task systems," *Technical Report 2013–008,* no. 2013.

[191] I. Lupu, P. Courbin, L. George, and J. Goossens, "Multi-criteria evaluation of partitioning schemes for real-time systems," in *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, 2010, pp. 1-8.

[192] A. Alahmadi, A. Alnowiser, M. M. Zhu, D. Che, and P. Ghodous, "Enhanced first-fit decreasing algorithm for energy-aware job scheduling in cloud," in *2014 International Conference on Computational Science and Computational Intelligence*, 2014, pp. 69-74.

[193] Z. Ren, T. Lu, X. Wang, W. Guo, G. Liu, and S. Chang, "Resource scheduling for delay-sensitive application in three-layer fog-to-cloud architecture," no.

[194] A. Cervin, B. Lincoln, J. Eker, K.-E. Arzén, and G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*, 2004, pp. 1-10.

[195] M. V. Micea, V.-I. Cretu, and V. Groza, "Maximum predictability in signal interactions with HARETICK kernel," *IEEE transactions on instrumentation and measurement,* vol. 55, no. 4, pp. 1317-1330, 2006.

[196] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM),* vol. 20, no. 1, pp. 46-61, 1973.

[197] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications* vol. 24: Springer Science & Business Media, 2011, ISBN: 1461406765.

[198] B. Rieck, "Basic analysis of bin-packing heuristics," *Publicado por Interdisciplinary Center for Scientific Computing. Heildelberg University,* no. 2010.

[199] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology,* vol. 12, no. 1, pp. 161-166, 2011.

[200] F. Fioretto, W. Yeoh, and E. Pontelli, "A multiagent system approach to scheduling devices in smart homes," in *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, 2017, pp. 240-246.

[201] R. S. Oliver, S. S. Craciunas, and G. Stöger, "Analysis of deterministic ethernet scheduling for the industrial internet of things," in *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2014, pp. 320-324.

LISTĂ LUCRĂRI PUBLICATE

[A1]. **Capota, E. A.**, Stangaciu, C. S., Micea, M. V., & Cretu, V. I. (2018, May). P_FENP: A Multiprocessor Real-Time Scheduling Algorithm. In *2018 IEEE 12th International Symposium on Applied Computational Intelligence and Informatics (SACI)* (pp. 000509-000514). IEEE.

Citat de:

- Boukir, K. (2020). *Mise en oeuvre de politiques d'ordonnancement temps réel multiprocesseur prouvée* (Doctoral dissertation, Nantes).
- Hanafi, M. E., Abozied, M. A., Elhalwagy, Y. Z., & Elfarouk, A. O. (2020, July). Real Time Symmetric Multiprocessing Software Design for Onboard Computer of Guided Missiles. In *2020 12th International Conference on Electrical Engineering (ICEENG)* (pp. 350-355). IEEE.

[A2]. **Capota, E. A.**, Stangaciu, C. S., Micea, M. V., & Curiac, D. I. (2019). Towards mixed criticality task scheduling in cyber physical systems: Challenges and perspectives. *Journal of systems and software*, *156*, 204-216.

Citat de:

- Riegler, M., & Sametinger, J. (2020, September). Mode Switching from a Security Perspective: First Findings of a Systematic Literature Review. In *International Conference on Database and Expert Systems Applications* (pp. 63-73). Springer, Cham.
- Al-Tarawneh, M. A. (2021). Bi-objective optimization of application placement in fog computing environments. *Journal of Ambient Intelligence and Humanized Computing*, 1-24.
- Padmajothi, V., & Iqbal, J. M. (2020). Adaptive neural fuzzy inference system-based scheduler for cyber–physical system. *Soft Computing*, *24*, 17309-17318.
- Chu, J., Zhao, T., Jiao, J., & Chen, Z. (2020). Optimal Design of Configuration Scheme for Integrated Modular Avionics Systems With Functional Redundancy Requirements. *IEEE Systems Journal*.
- Amarnath, A., Pal, S., Kassa, H. T., Vega, A., Buyuktosunoglu, A., Franke, H., ... & Bose, P. (2021). Heterogeneity-Aware Scheduling on SoCs for Autonomous Vehicles. *IEEE Computer Architecture Letters*.
- JIANG Baihua, LYU Xuefeng, LIU Yulong. Realization of cyber-physical systems for smart petrochemical factory based on agents[J].CIESC Journal, 2021, 72(3): 1575-1584.
- Martsenyuk, V. P., & Sverstyuk, A. S. (2020). Mathematical models and methods for compartmental modeling of cyber physical systems in medical and biological processes.
- Sverstyuk, AS, & Sverstyuk, AS (2020). Models and methods for compartmental mathematical modeling of cyber physical systems in medical and biological processes (Doctoral dissertation, TNTU named after I. Pulyuy).
- Fadlelseed, S., Kirner, R., & Menon, C. (2021). ATMP-CA: Optimising Mixed-Criticality Systems Considering Criticality Arithmetic. Electronics 2021, 10, 1352.
- Fadlelseed, S., Kirner, R., & Menon, C. (2021). ATMP-CA: Optimising Mixed-Criticality Systems Considering Criticality Arithmetic. *Electronics*, *10*(11), 1352.

- Shukalov, A. V., Zharinov, I. O., & Zharinov, O. O. (2020, November). Industrial cyber-machines remote control method. In *Journal of Physics: Conference Series* (Vol. 1661, No. 1, p. 012109). IOP Publishing.

[A3]. **Capota, E. A.**, Stangaciu, C. S., Micea, M. V., & Curiac, D. I. (2020). Towards Fully Jitterless Applications: Periodic Scheduling in Multiprocessor MCSs Using a Table-Driven Approach. *Applied Sciences*, *10*(19), 6702.

Lucrări trimise care sunt sub examinare:

[A4]. Stangaciu, C. S., **Capota, E. A.**, Micea, M. V., & Curiac, D. I. (2021). A Hardware-Aware Application Execution Model in Mixed Criticality Internet of Things.