METODE DE PLANIFICARE TIMP REAL PENTRU SISTEME DISTRIBUITE CU NIVELURI MIXTE DE CRITICALITATE

Teză de doctorat – Rezumat

pentru obținerea titlului științific de doctor la
Universitatea Politehnica Timișoara
în domeniul de doctorat Calculatoare și Tehnologia Informației

autor ing. Eugenia Ana CAPOTA
conducător științific Prof.univ.dr.habil.ing. Mihai V. MICEA

noiembrie 2021

Cuprins:

1. Introducere

1.1 Domeniile abordare

Sistemele de timp real (RT: Real-Time) sunt prezente în viața de zi cu zi, fiind utilizate în numeroase domenii. 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 de comunicare” [1].

Multe aplicații de timp real critice au fost deja implementate folosind arhitecturi distribuite [2]. Aceste arhitecturi conțin o colecție de componente independente, împărțite pe mai multe unități de procesare și care comunică între ele prin intermediul unei rețele [3].

Un domeniu vast care utilizează arhitecturi distribuite este reprezentat de sistemele cyber physical (CPSs: Cyber Physical Systems). 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”* [4].

Un alt domeniu care extinde aplicabilitatea sistemelor distribuite eterogene este Internet of Things (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 [5].

În această teză accentul se va pune pe metode de planificare pentru sisteme cu niveluri mixte de criticalitate.

1.2 Obiectivele tezei

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, cercetarea realizată a presupus îndeplinirea următoarelor obiective, fiecare obiectiv fiind împărțit în mai multe etape:

[O1]. Studiul și analiza stadiului actual al domeniului sistemelor cu niveluri mixte de criticalitate, cu accent pe sistemele distribuite.
- *[T1.1]. Compararea diferitelor modele de task-uri existente.*
- *[T1.2]. Clasificarea algoritmilor de planificare principali în funcție de arhitectura platformei pe care rulează.*

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

[O3]. Dezvoltarea unui mecanism de planificare.
- *[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.*

[O4]. Testarea și validarea algoritmului.
- *[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.

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, ale sistemelor cyber physical. Provocările și avantajele integrării conceptului de niveluri mixte de criticalitate în sistemele cyber physical sunt, de asemenea, evidențiate. Următoarea secțiune analizează aspectele teoretice din lucrările de specialitate ce abordează domeniul IoT și propune o arhitectură MC-IoT pentru sisteme de

timp real distribuite cu niveluri mixte de criticalitate.

Capitolul 4 definește un nou model de task-uri pentru sisteme distribuite cu niveluri mixte de criticalitate. Acesta permite o administrare cât mai eficientă a resurselor platformei. 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.

Capitolul 5 prezintă mediul de simulare adaptat pentru sisteme eterogene distribuite cu niveluri mixte de criticalitate.

Capitolul 6 evaluează tehnicile de stabilire a afinității, precum și metodologia de mapare a task-urilor prin implementarea și testarea acestora în cadrul simulatorului.

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.

Capitolul 8 propune 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.

Capitolul 9 realizează 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.

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

2. Stadiul actual în domeniul sistemelor cu niveluri mixte de criticalitate

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. Vestal a propus primul model de execuție al task-urilor pentru MCSs [6]:

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

unde:

- l este numărul de niveluri de criticalitate;
- T_i reprezintă intervalul (minim) de timp între două instanțe consecutive ale aceluiași task i;
- D_i indică momentul până la care trebuie încetată execuția unei instanțe, relativ la timpul de activare;
- L_i este nivelul de criticalitate;
- 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).

Pe baza acestui model de task-uri și a altor extensii au fost dezvoltați numeroși algoritmi de planificare. 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 [7]:

- 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.

Pentru sistemele cu mai multe unități de procesare există patru categorii de algoritmi de planificare [3, 7-9]:

- Clasa P: Algoritmi de planificare partiționați – fiecare task este alocat unei singure unități de procesare.
- Clasa G: Algoritmi de planificare globali – permit migrarea task-urilor între unitățile de procesare.
- Clasa C: Algoritmi de planificare cluster/semi-partiționați – abordare hibridă între algoritmii partiționați și cei globali în care unitățile de procesare sunt grupate în clustere și sub-clustere.
- Clasa D: Algoritmi de planificare distribuiți – folosesc rețele de comunicare pentru a interconecta partiții, fiecare partiție fiind alcătuită din unul sau mai multe procesoare.

3. Sisteme distribuite

3.1 Sisteme cyber physical

CPSs încapsulează numeroase sisteme cu tehnologii avansate din domenii precum: automotive, avionică, dispozitive medicale, platforme industriale, etc. 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 [10]:

- **Eterogenitatea** – se referă la specificații hardware variate, precum și diferite cerințe ce țin de aplicație sau de consumul de putere.
- **Administrarea eficientă a puterii** – trebuie avută în vedere mai ales în cazul componentelor ce folosesc surse de alimentare sub formă de baterii.
- **Dinamism și adaptabilitate** – pentru interacțiunea directă cu mediul înconjurător, care este de multe ori dinamic și imprevizibil.
- **Robustețe** – aici este de preferat ca aplicațiile critice să nu fie afectate de condițiile incerte ale mediului când vine vorba de comportamentul în timpul rulării sistemului sau de overload-ul computațional cauzat de alte aplicații.
- **Distribuție** – se referă la algoritmii de planificare pentru sisteme cu mai multe unități de procesare care țin cont de locația componentelor și de comunicarea dintre acestea.
- **Scalabilitate** – atunci când sunt utilizate sisteme cu mai multe unități de procesare, scalabilitatea acestora în termeni de model, mecanisme de planificare, componente hardware, analiză și testare este foarte importantă.
- **Securitate și izolare** – securitatea este mai greu de asigurat într-un sistem care conține funcționalități de criticalitate diferită. Când vine vorba de politicile de planificare, trebuie asigura izolarea între aplicațiile critice și cele care nu sunt critice, atât din punctul de vedere al comportamentului temporal, cât și a utilizării resurselor.

Conform acestor caracteristici, în teză sunt prezentate o serie de atribute (Tabel 1) pe baza cărora pot fi evaluați algoritmii de planificare pentru MCSs.

Tabel 1. 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

3.2 Internet of Things

IoT a fost aplicat pe scară largă în numeroase domenii, precum cel medical [11] sau sisteme industriale [12, 13]. Astfel, diferite tipuri de arhitecturi fizice pentru IoT au fost propuse și descrise în literatura de specialitate, dintre acestea arhitectura bazată pe Fog este cea mai potrivită pentru aplicațiile de timp real distribuite complexe. Arhitecturile bazate pe Fog sunt alcătuite din trei niveluri [14]: 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).

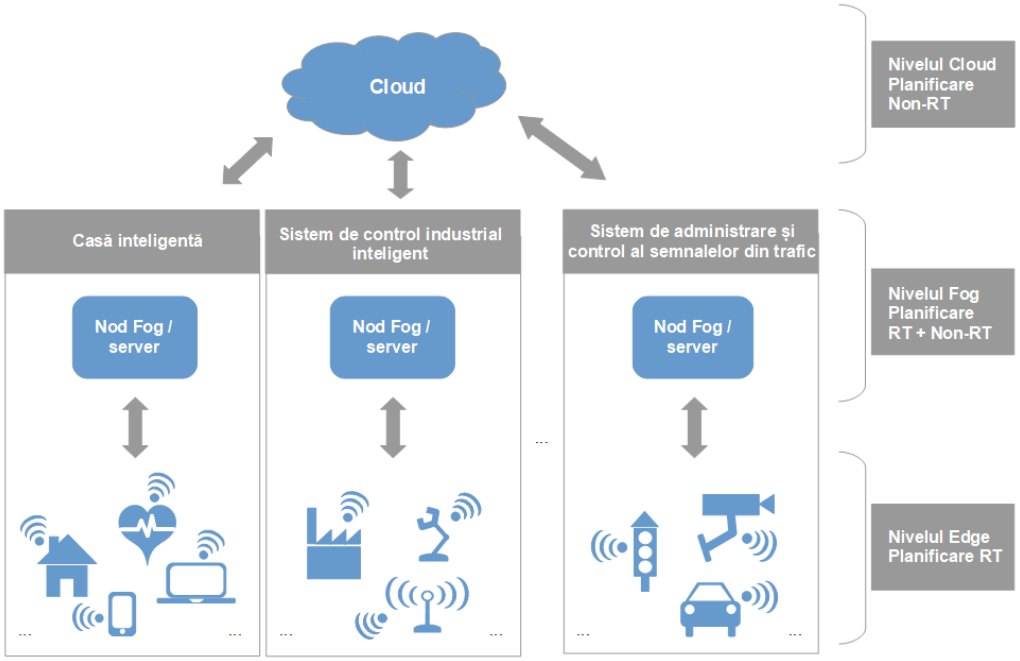

Figura 1. Arhitectura Fog MC-IoT propusă.

În Figura 1 sunt considerate una sau mai multe rețele Fog centralizate eterogene conectate la nivelul Cloud. Fiecare rețea are un nod Fog care se ocupă cu procesarea și stocarea datelor primite de la dispozitivele Edge din rețeaua respectivă. Nodurile Edge pot comunica între ele, dar și cu nodurile Fog, și sunt reprezentate de către orice tip de dispozitiv capabil de procesare.

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. În această arhitectură fiecare element de procesare reprezintă un sistem cu o singură unitate de procesare.

Pe baza arhitecturii descrise, în cele ce urmează este prezentat un model de execuție al task-urilor și o formalizare matematică a problemei de planificare.

Adaptarea conceptului de sisteme cu niveluri mixte de criticalitate în arhitecturile IoT, dă naștere unei noi paradigme, denumită MC-IoT, pentru care această teză 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.*

Problema planificării la nivelul Edge poate fi împărțită în două subprobleme:

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.

4. O metodologie de mapare a task-urilor pe diferite elemente de procesare

Pornind de la modelul clasic pentru sisteme cu niveluri mixte de criticalitate definit de Vestal în 2007 [6] 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.

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.

Folosind modelul de task-uri definit anterior, precum și problema planificării formalizată în Secțiunea 3.2, 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 pe baza timpului de execuție:

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 $q \leq p$.

4.3 Setarea scorului de afinitate pe baza nivelului de criticalitate:

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

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.

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

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, unde q variază de la 1 la $p - 1$.

4.4 Maparea task-urilor

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.

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 și se verifică dacă utilizarea elementului de procesare este mai mică sau egală decât 1 pentru toate nivelurile de criticalitate.

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

Algoritmul de mapare a task-urilor Best Affinity Fit (BAF) a fost, inițial, implementat în MATLAB, pentru testarea sa și compararea cu cele două metode: 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 [15]. 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 2. Generarea seturilor de task-uri s-a realizat în MATLAB. Toate task-urile au fost generate aleatoriu, utilizând algoritmul introdus în [16] și care are la bază metoda lui Guan [17]. Interfața grafică pentru vizualizarea planificării task-urilor a fost dezvoltată tot în MATLAB.

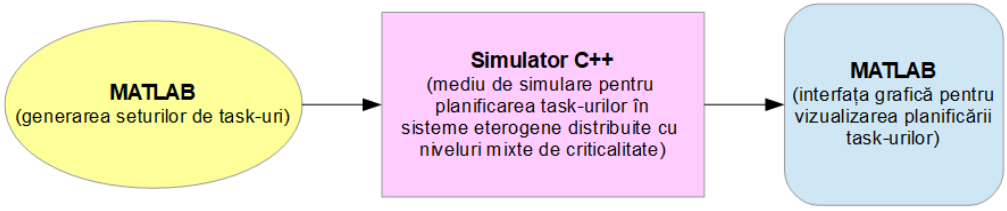

Figura 2. Modul de funcționare al mediului de simulare.

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) [18-20]. 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 $\{Lo, Hi\}$. Fiecare punct de date de pe grafic a fost determinat prin generarea aleatorie a 1000 de seturi de task-uri.

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.

BAF vs. BFDC: Valorile scorului de afinitate au fost atribuite în funcție de nivelul de criticalitate al fiecărui task (Figura 3).

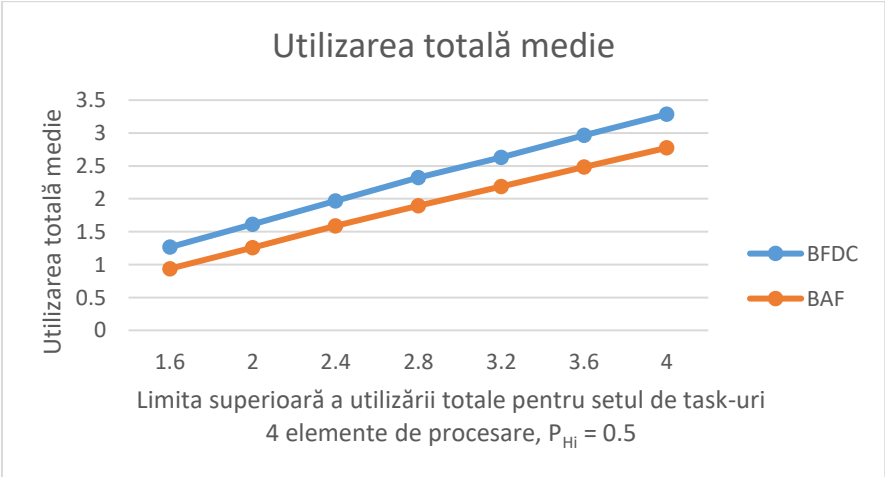

Figura 3. Utilizarea totală medie a elementelor de procesare, obținută prin variația limitei superioare a utilizării totale pentru fiecare set de task-uri.
$\boldsymbol{U_L = 0.05, U_U = 0.75, Z_L = 1, Z_U = 8}$.

Pentru ambele seturi de experimente, algoritmul de mapare Best Affinity Fit a prezentat o performanță mai bună decât BFDU și BFDC, din punctul de vedere al utilizării totale medie.

Deviația medie a scorului de afinitate pentru seturile de task-uri generate în Figura 3 este reprezentată în Tabel 2.

Tabel 2. Valorile pentru deviația medie a scorului de afinitate.

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.

Modelul de task-uri utilizat în teză are la bază modelului de task-uri periodice în timp real introdus în [21]. 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:

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

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;
- 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 [22]. Astfel, job-ul k al task-ului M_i este caracterizat de:

$$J_{i,k} = \{a_{i,k}, d_{i,k}, c_{i,k}, s_{i,k}, T_i, D_i, L_i\} \tag{7-2}$$

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 (7-1).

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:*

$$s_{i,1} - s_{i,0} = s_{i,2} - s_{i,1} = \cdots = s_{i,n} - s_{i,n-1} = T_i \tag{7-3}$$

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 [21].

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 (7-1), astfel:

$$M_i \equiv \left\{T_i, D_i, L_i, \{C_{i,L_j} \mid j \in 1 \dots l\}, \{S_{i,L_j} \mid j \in 1 \dots l\}\right\}, \quad \text{unde pentru orice task } k,\ T_k \leq T_i \text{ pentru } k < i \tag{7-4}$$

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.*

Testul de fezabilitate rulează în modul următor: task-urile din set sunt ordonate crescător în funcție de perioadă. În continuare sunt extrase câte 2 task-uri pentru care se calculează cel mai mare divizor comun. Dacă suma WCET-urilor pentru cele 2 task-uri este mai mare decât cel mai mare divizor comun pentru un anumit mod de rulare al sistemului, atunci testul de fezabilitate este negativ. Dacă, în schimb, toate task-urile au fost verificate cu succes, testul de fezabilitate este pozitiv.

În continuare este propusă o adaptare pentru MCSs a algoritmului de planificare în timp real, table-driven, FENP [21] pentru sisteme cu o singură unitate de procesare, precum și a variantei sale partiționată, P_FENP [23] pentru sisteme cu mai multe unități de procesare.

Algoritmul de planificare FENP_MC creează, în etapa offline, un tabel al planificării pentru fiecare nivel de criticalitate al sistemului pe baza testului de fezabilitate propus în [21] pentru sisteme de timp real, și care a fost adaptat și prezentat în această teză pentru sisteme cu niveluri mixte de criticalitate.

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

$$\Gamma_q = \{TaskID;\ StartTime\} \tag{7-5}$$

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

8. P_FENP_MC: Partitioned Fixed Execution Non-Preemptive Mixed Criticality

P_FENP_MC constă din două etape, o etapă offline (Figura 4) și una online (Figura 5). 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.

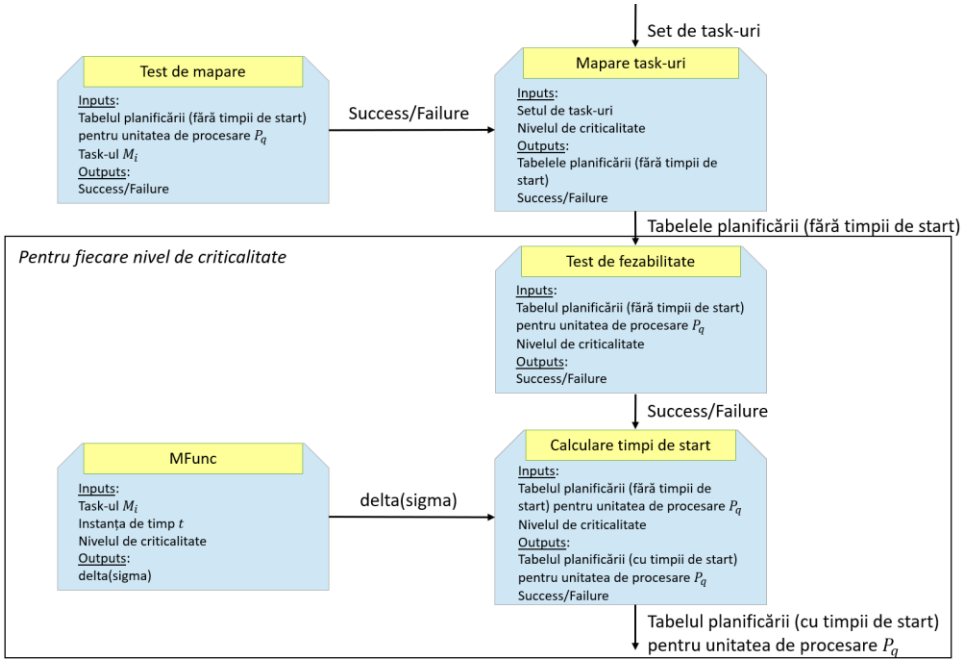

Figura 4. Execuția etapei offline.

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 [24]:

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

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 unități de procesare, unde se va executa același test.

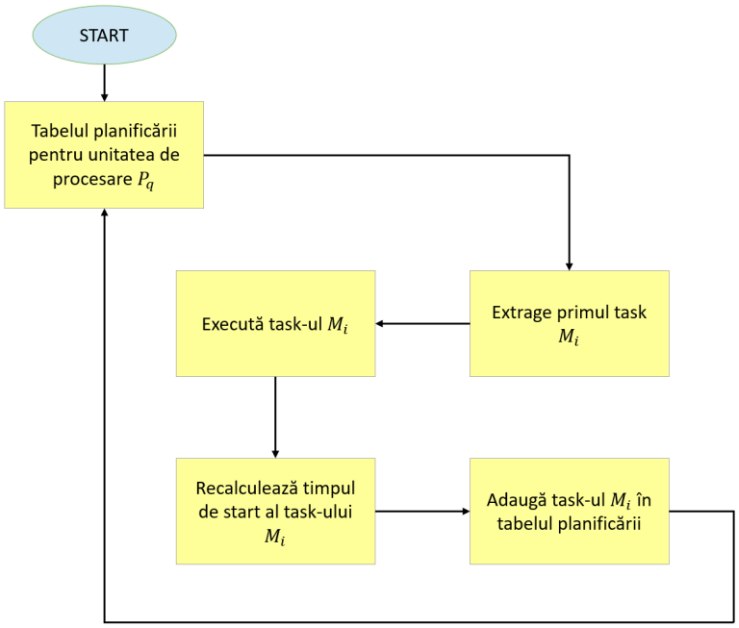

Figura 5. Execuția etapei online.

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 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, 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.).

9. Evaluarea performanței

Acest capitol 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) [25], dar și cu o extensie dezvoltată pentru task-uri periodice pe baza algoritmului table-driven introdus în [26], 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) [27], 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) [28], dar și cu metodele event-driven și table-driven descrise anterior. Aceste simulări au fost executate într-un context non-preemptiv, în cadrul unui sistem omogen, cu două niveluri de criticalitate, utilizând mediul de simulare descris anterior.

9.2 Rata de succes

În cazul algoritmului P_FENP_MC, cu cât numărul de unități de procesare este mai mare, 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 (Capitolul 8). 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 [29]. 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 [26, 28].

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.

9.3 Execuție fără jitter – Test Case

Jitter-ul unui task este calculat ca fiind diferența dintre separarea maximă și minimă între două job-uri consecutive ale aceluiași task τ_i [30]:

$$Jitter(\tau_i) = \underset{k\geq 1}{max}\{|\, s_{i,k} - s_{i,k+1}|\} - \underset{k\geq 1}{min}\{|\, s_{i,k} - s_{i,k+1}|\} \qquad (9\text{-}1)$$

unde:

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

Tabel 3 conține valorile jitter-ului, obținute aplicând expresia (9-1), pentru un exemplu de set cu trei task-uri planificat utilizând P_FENP_MC, P-EDF-VD (varianta non-preemptivă), TT-Merge, Energy-efficient TT-Merge și P-TT-OCBP.

Tabel 3. Valorile jitter-ului pentru un exemplu de set cu trei task-uri 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 perspective

Principalele contribuții aduse în această teză sunt:

- Identificarea unor caracteristici comune ale CPSs și evaluarea algoritmilor de planificare pentru MCSs din literatura de specialitate, pe baza acestor atribute.
- Propunerea unei arhitecturi IoT cu niveluri mixte de criticalitate (MC-IoT).
- Definirea unui model de task-uri pentru sisteme de timp real distribuite cu niveluri mixte de criticalitate și introducerea unei metodologii de mapare a task-urilor.
- Implementarea unui algoritm de planificare perfect periodică pentru sisteme de timp real cu niveluri mixte de criticalitate.
- Îmbunătățirea mediului de simulare introdus în [15] pentru sisteme omogene cu niveluri mixte de criticalitate din literatura de specialitate astfel încât să fie utilizat pentru sisteme eterogene distribuite cu niveluri mixte de criticalitate.

Ca direcție de dezvoltare se urmărește:

- Implementarea practică a algoritmului de planificare, pe baza modelului de task-uri introdus, utilizând LITMUS-RT.
- Extinderea algoritmului de planificare pentru sisteme cu mai mult de două niveluri de criticalitate.
- Execuția task-urilor de criticalitate scăzută chiar și după trecerea de la un nivel de criticalitate la altul.
- Implementa unei metode care să permită revenirea sistemului în modul de rulare inițial, după o trecere la un nivel de criticalitate ridicat, 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 – utilizează în mare parte platforme distribuite, cu niveluri mixte de criticalitate. Astfel, algoritmul de planificare introdus 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 – 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.
- IoT – algoritmul poate fi aplicat 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] 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.

[3] 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.

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

[5] 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.

[6] 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.
[7] 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.
[8] 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.
[9] 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.
[10] 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.
[11] 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.
[12] D. B. Rawat, C. Brecher, H. Song, and S. Jeschke, *Industrial Internet of Things: Cybermanufacturing Systems*: Springer, 2017, ISBN: 3319425587.
[13] 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.
[14] 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.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] 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.
[21] 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.
[22] 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.
[23] 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.
[24] D. Socci, "Scheduling of certifiable mixed-criticality systems," Grenoble Alpes, 2016.
[25] 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.
[26] 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.
[27] B. Rieck, "Basic analysis of bin-packing heuristics," *Publicado por Interdisciplinary Center for Scientific Computing. Heildelberg University*, no. 2010.
[28] 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.
[29] 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.
[30] 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.