Algoritm îmbunătățit de planificare a sarcinilor executate în sisteme de tip grid

Ovidiu Gherman, Ştefan Gheorghe Pentiuc

***Abstract*—Job scheduling in grid systems is a vital component of a grid infrastructure composed mostly of heterogeneous elements. The algorithms used must have some stringent specifications regarding speed, resource allocation efficiency and adaptation to a dynamic environment especially when considering quality of service requirements. Equilibrium must be obtained – between the user satisfaction (that ultimately is proof of system's usability) and the service provider's desire to use efficiently the hardware and software infrastructure.**

***Index Terms*—grid scheduler, grid simulator, QoS, scheduling algorithm.**

I. INTRODUCERE

Planificarea sarcinilor pentru execuția pe infrastructuri de tip grid reprezintă o componentă importantă a sistemelor de calcul paralel şi distribuit din ziua de azi. Acest lucru permite o utilizare eficientă a resurselor hardware şi software limitate, în timp ce se oferă capabilități de asigurare a calității serviciilor non-triviale pentru utilizatori.

Scopul principal al unui planificator este acela de a descoperi resursele disponibile în cadrul platformei grid (este necesară cunoaşterea specificațiilor şi a politicilor locale a fiecărui nod), de a crea un plan pentru distribuția sarcinilor către resursele disponibile şi de a asigura maparea sarcinilor primite – conform planului – pe resursele corespunzătoare, în scopul folosirii eficiente a respectivelor resurse cu păstrarea calității serviciilor către utilizator. Din punctul de vedere al utilizatorului, acesta foloseşte în mod transparent un intermediar de resurse care va descoperi resursele disponibile şi compatibile cu cerințele aplicației rulate şi va negocia anumite servicii (cantitativ şi calitativ) conform costurilor şi necesităților respectivului utilizator. Acelaşi intermediar, folosind servicii specifice, este responsabil cu maparea sarcinilor primite (în mod static sau dinamic) de la utilizator/utilizatori pe resursele planificate, cu lansarea aplicației precum şi cu strângerea rezultatelor şi transmiterea către utilizator într-un format acceptat. De asemenea, serviciul de planificare trebuie să fie suficient de rapid încât să nu introducă întârzieri în execuția sarcinilor şi să nu consume o cantitate mare de resurse pentru executarea calculelor necesare (caz în care poate apare un overhead semnificativ pentru planificatoarele complexe).

Complexitatea problemei este ridicată pentru sarcini de lucru de mari dimensiuni (număr mare de sarcini, cantitate ridicată de resurse de gestionat), astfel încât algoritmii folosiți în sisteme reale caută o soluție sub-optimală suficient de bună pentru a rezolva problema fără a introduce întârzieri semnificative în execuția sarcinilor. Introducerea unor constrângeri de asigurarea a calității serviciilor poate duce la soluții chiar mai ineficiente, o dată cu creşterea numărului acestor parametri. Astfel, optimizarea planului de execuție final este de dorit, în special în cazul condițiilor dinamice din grid (mai ales dacă luăm în considerare faptul că o caracteristică importantă a sistemelor grid este lipsa unui control total, centralizat, asupra resurselor gridului).

II. PLANIFICATOARE GRID

Gridurile computaționale reprezintă sisteme dinamice care pun probleme importante pentru planificarea sarcinilor datorită naturii lor eterogene şi distribuite. De obicei este necesară menținerea unui anumit nivel de performanță, ținând cont de cerințele de calitate a serviciilor şi de siguranța/robustețea sistemului (hardware şi în special software).

Planificarea sarcinilor poate avea – de obicei - unul din cele doua scopuri: calculele de înaltă performanță (se încearcă minimizarea timpului de execuție în programarea paralela sau HPC) sau execuția a cât mai multor sarcini care să se finalizeze cu succes, pe unitatea de timp (se doreşte asigurarea unei disponibilități crescute a resurselor şi atingerii parametrilor de calitate predefiniți – mai ales în cazul sistemelor de uz general); în acest caz, planificatorul trebuie să decidă ordinea şi locul de execuție al sarcinilor cu îndeplinirea anumitor criterii – de obicei criterii de calitate a serviciilor.

Maparea sarcinilor şi restul deciziilor de planificare pot fi realizate dinamic, în momentul primirii sarcinii (mod on-line sau planificare dinamică), sau atunci când un număr predefinit de sarcini (sau toate) a fost recepționat (modul *batch*). Dacă în primul caz planificatorul permite mai multă flexibilitate şi un timp de răspuns scăzut, în cel de-al doilea caz se poate realiza o planificare mai eficientă. Replanificarea dinamică pentru

Manuscris trimis pe 6 decembrie 2010.

O. Gherman este doctorand în domeniul Calculatoare şi Tehnologia Informației în cadrul Universității "Ştefan cel Mare" Suceava, , str. Universității nr. 13 (tel: 0741/912.395; e-mail: ovidiuh@stud.usv.ro).

Ş. G. Pentiuc este profesor în cadrul Universității "Ştefan cel Mare" Suceava, str. Universității nr. 13 (e-mail: pentiuc@eed.usv.ro).

fiecare nouă sarcină este mai puțin fezabilă din cauza necesarului computațional (timp de răspuns crescut din partea planificatorului şi resurse ocupate pentru sistemul de management), totuşi aduce avantaje certe pentru sisteme distribuite eterogene [1].

O schemă de alocare a resurselor poate fi definită astfel: dacă avem un set T de n sarcini independente (cu aceeaşi lungime) şi m procesoare în cadrul unui sistem grid, schema de alocare S a setului T pe m resurse este un set definit de tupla triplă (v,p,t). O asemenea tuplă aparținând setului S defineşte faptul că procesorul p prelucrează sarcina v între momentul t şi $t+d$ (acesta din urmă reprezentând timpul de finalizare), fiecare sarcină v fiind executată cel puțin o dată, iar fiecare procesor p poate executa cel mult o sarcină la un moment dat. Diferența între timpul de finalizare şi timpul de start îl reprezintă timpul de execuție. S-a notat cu v o sarcină din setul T, cu p indexul procesorului (care face parte din m) iar cu t timpul de start al sarcinii v. O asemenea tuplă triplă reprezintă o instanță a unei sarcini [2], în care sarcina poate fi asignată pentru mai mult de un singur procesor.

Planificatoarele au eficiență ridicată mai ales pentru probleme statice (se cunosc sarcinile şi necesitățile acestora în avans). Un asemenea exemplu de sistem de management ce permite rezervarea în avans a sarcinilor este GORBA (care foloseşte politici simple de planificare şi un algoritm evolutiv pentru determinarea soluției sub-optime) şi CCS (un hibrid ce foloseşte în paralel mai multe cozi de sarcini FCFS, optimizate prin tehnici de backfilling).

Planificatoarele (spre deosebire de cozile de execuție) realizează de asemenea – implicit sau explicit – rezervarea resurselor în avans. Acesta lucru, singur, impune un anumit nivel de calitate a serviciilor (utilizatorul cunoaşte că sarcina sa a fost acceptată şi planificată, cu mari şanse să fie executată cu succes). Pentru impunerea unor restricții mai puternice, se pot implementa mecanisme care să asigure respectiva rezervare (de exemplu, în cazul pierderii acesteia în urma unei erori hardware/software sarcina poate fi migrată pe altă resursă), la nivelul arhitecturii gridului sau chiar la nivelul algoritmului de planificare, folosind agenți specializați. Aceştia permit să se specifice când şi pe ce maşini vor fi rulate sarcinile (care vor fi cunoscute în avans). Acest lucru permite inclusiv rezervarea în avans a resurselor, re-maparea dinamica a sarcinilor pe resurse mai potrivite şi optimizarea planificării pe baza diferitelor metode de optimizare (cum ar fi metodele de căutare locală). Un plan de execuție permite să se specifice pe ce maşină şi când va fi executată o anumită sarcină, astfel încât această informație va fi cunoscută în avans (rezervarea resurselor se face în avans).

Planificatoarele folosite în sistemele grid pentru managementul resurselor pot fi centralizate, ierarhice sau descentralizate (cu anumite avantaje şi dezavantaje) [3], [4]. Planificatoarele ierarhice se mai numesc şi hibride (folosesc caracteristici diferite de ale celor două tipuri de modele).

Abordarea bazată pe planificatori este de obicei mai eficientă decât cea bazată pe cozi de execuție [5]. Câteva dezavantaje sunt totuşi notabile:

- pentru a fi eficient, un serviciu de planificare necesită o sumă suplimentară de informații referitoare la sarcinile primite pentru execuție (necesități de rulare, timpi estimați de execuție, capabilitățile resurselor, etc.); cu cât aceştia sunt mai precişi, cu atât este mai eficient algoritmul de planificare;
- serviciile de planificare lucrează mai bine în cazul problemelor statice (număr fix de sarcini de lucru);
- deseori se lucrează cu estimări privind diverşi timpi relevanți pentru sarcinile supuse planificării, dacă acestea sunt exagerate (în sens negativ sau pozitiv) durata de execuție a planului compilat (*makespan*) poate creşte, ceea ce duce, de obicei, la o recalculare completă sau parțială, lucru care poate fi costisitor din punct de vedere computațional.

A. Asigurarea calității serviciilor prin planificarea sarcinilor

Un furnizor de servicii va ajunge de obicei la un consens cu utilizatorul în ceea ce priveşte calitatea serviciilor printr-un acord SLA (Service Level Agreement) care va specifica condițiile serviciilor oferite, costul acestora precum şi metrica QoS care trebuie să fie urmată de către furnizor (tipul acesteia poate fi selectat de client – dacă furnizorul permite aceasta – şi poate influența costul resurselor alocate). Metricile tipice sunt la nivelul serviciilor (adică utilizatorul va specifica un anumit scop, cum ar fi un termen limită care trebuie sa fie respectat, fără sa cunoască caracteristicile fizice ale platformei permițând furnizorului să aloce dinamic resursele [6].

În cazul sistemelor necomerciale, chiar dacă cerințele nu sunt atât de stringente, specificațiile QoS trebuie să țină cont de utilizatori şi de prioritatea sarcinilor (anumiți utilizatori pot avea permisiunea de a accesa mai multe resurse şi drepturi de execuție preferențiale) sau de anumite cerințe speciale (cum ar fi utilizarea unei platforme specifice).

Una din problemele folosirii SLA este că în anumite sisteme în care metrica se bazează pe caracteristicile fizice (de exemplu un anumit număr de cicluri CPU sunt alocate sau se aşteaptă un anumit nivel de disponibilitate din partea resursei) o parte din resurse sunt blocate prin supra-alocare (resurse puse în aşteptare fie ca şi rezervă sau nefolosite din cauza algoritmilor de alocare ineficienți sau a estimărilor defectuoase cu privire la durata de execuție) şi nu pot fi folosite de către alte sarcini (aparținând altor utilizatori) pentru ca va încălca SLA-ul din partea furnizorului (respectivele resurse fizice fiind alocate exclusiv către client). Pentru a evita asemenea dezavantaje, sistemele necomerciale pot folosi o ierarhizare pe baza de reputație pentru resurse şi pentru a clasifica utilizatorii pe nivele predefinite care vor face mai uşoară planificarea sarcinilor [7].

Un pas important în tehnicile de planificare moderne îl reprezintă precizia estimării timpului de realizare a sarcinii. De obicei, acest lucru este efectuat chiar de către utilizator (în baza analizei codului sursa şi a performantei relative – sau a performantei echivalente a platformei – a maşinilor-țintă). Există metode de a calcula automat timpul de execuție cu predicție pe termen scurt, care presupun ca planificatorul să

analizeze timpul de execuţie al sarcinilor rapide (cele care au de obicei o durata de câteva minute) şi apoi să corecteze estimările iniţiale (de exemplu cel folosit în AppLeS) sau modelul de predicţie pe termen lung (de exemplu GHS - Grid Harvest Service) [8].

Pentru îndeplinirea aşteptărilor QoS, se adaugă o componentă QoS specializată în algoritmul de planificare convenţional şi se foloseşte predicţia de performanţă cea mai recentă (în sistemele grid de uz general) pentru a se estima un timp de execuţie mai precis prin corecţii succesive.

În analiza unui algoritm de planificare, o importanţă critică o au funcţiile-obiectiv. Acestea permit stabilirea calităţii rezultatului algoritmului respectiv, generând o comparaţie corectă şi consistentă între diferiţi algoritmi.

Cea mai comună funcţie-obiectiv folosită în planificarea sarcinilor de lucru este timpul maxim de terminare al sarcinilor dintr-un set (*makespan*). Acesta are avantajul că ne furnizează informaţii cu privire la utilizarea eficientă a resurselor disponibile. Astfel, un timp maxim cât mai mic pentru un set de date *S* dat înseamnă că acel set a fost rulat pe un sistem într-un mod mai eficient, ”împachetând” mai strâns sarcinile disponibile şi menţinând sistemul mai ocupat, dar pe o perioadă mai scurtă de timp. În cazul sistemelor care folosesc predicţia duratei de execuţie din partea utilizatorului, această estimare este foarte importantă deoarece o estimare eronată poate duce la întârzieri în planificarea sarcinilor sau la sarcini care vor fi suspendate/oprite dacă depăşesc perioada de timp alocată. Calea cea mai evidentă de a îndeplini specificaţiile de calitate a serviciilor este utilizarea resurselor în avans (conform planificării curente) şi confirmarea rezervării până în momentul utilizării efective ale resursei respective.

Aceşti parametri ai planificatorului pot fi consideraţi ca definind calitatea gridului atunci când sunt aplicaţi unui set de date de intrare sau consideraţi din punct de vedere al istoricului sistemului (pe termen lung pot indica fiabilitatea sistemului şi nivelul de satisfacere al calităţii serviciilor către utilizatori). Dar tot ei pot fi luaţi în considerare pe termen scurt, atunci când sunt folosiţi pentru a alege o planificare nouă, mai eficientă decât cea anterioară. Aceşti parametri pot deveni factori de selecţie obiectivi pentru stabilirea calităţii noului plan de execuţie. Astfel, dacă avem două planuri de execuţie – unul stabilit anterior ca fiind optim şi unul candidat pentru verificarea calităţii, putem folosi criteriul termenului maxim de execuţie al setului de sarcini (în cazul planificatoarelor dinamice acesta este setul de date care începe cu prima sarcină primită luată în considerare şi se termină cu sarcina primită în mod curent; sistemul acceptă şi noi sarcini care vor fi planificate la sosire) pentru stabilirea calităţii noului plan de execuţie prin prisma consumului eficient de resurse. Astfel se poate scrie relaţia

$$M = \frac{makespan_{best} - makespan_{candidat}}{makespan_{best}} \quad (1)$$

De obicei acest lucru este insuficient dacă dorim să luăm în considerare şi calitatea serviciilor oferite de grid. În acest caz, vom considera un alt parametru, şi anume durata de execuţie a sarcinii înainte de expirarea termenului limită. Dacă vom considera sarcinile care nu sunt întârziate, vom nota numărul acestora astfel

$$D = \frac{notdelayed_{candidat} - notdelayed_{best}}{notdelayed_{best}} \quad (2)$$

B. Algoritmi de planificare a sarcinilor în grid

Algoritmii de planificare orientaţi spre implementarea constrângerilor QoS iau în considerare metricile relevante în funcţie de natura respectivelor constrângeri, aşa cum au fost definite de administratorii sistemului. În abordările de tip ”best effort” factorii de calitate a serviciilor nu influenţează procesul deoarece nu se dau nici un fel de garanţii. Pentru oferirea unui modul de asigurare a calităţi serviciilor oferite de grid (de nivel non-trivial) este nevoie ca anumite scopuri să fie impuse chiar la momentul proiectării algoritmului de distribuţie a sarcinilor. De obicei, pentru sistemele de uz general, acest scop este cel de a avea un raport sarcini rezolvate cu succes / total sarcini primite cât mai mare (după ce se exclud sarcinile incompatibile – cele care au un necesar de resurse indisponibil fizic, care au necesităţi speciale ce nu pot fi îndeplinite din lipsă de resurse sau cele ce conţin programe cu erori sau cu parametri incomplet specificaţi – dacă sunt ceruţi).

Algoritmul FCFS (First Come, First Served) procesează sarcinile de intrare în ordinea sosirii. Acestea sunt preluate de utilizator şi adăugate în coada de aşteptare a proceselor. O sarcină va aştepta – pentru a fi executată – execuţia celorlalte sarcini din coadă, situate în faţa sa. Nu se asignează nici o prioritate sarcinilor sosite, nu se iau în considerare situaţii adiţionale (de exemplu starea resurselor, etc.). Planificatorul MAUI este unul din utilizatorii acestui algoritm.

O altă abordare care poate fi folosită în cazul unei cozi de sarcini este cea de tip backfilling. În principiu, acest lucru înseamnă reordonarea unei cozi (creată iniţial printr-un algoritm de tip FCFS, deşi se pretează oricărui tip de coadă, ca metodă de optimizare ulterioară). Dacă mai multe sarcini rulează simultan (necesităţile lor de resurse de procesare, cumulate, sunt egale sau sub cele ale gridului utilizat) iar următoarea sarcină din coadă are nevoie de toate procesoarele unui sistem, un algoritm FCFS va rezerva procesoarele eliberate din setul curent şi le va pune în stare de aşteptare pentru a fi gata să lanseze procesul imediat următor din coadă. Până la epuizarea tuturor sarcinilor curente, respectivele resurse vor fi în starea idle, fără a face calcul util (risipă de resurse). Un algoritm ce foloseşte o metodă de backfilling va umple respectivele sloturi disponibile (resurse în aşteptare) cu alte sarcini, mai mici (care necesită un număr mai mic de procesoare), aflate mai departe în coadă decât sarcina curentă, sarcini care se pretează la a rula pe resursele eliberate (numărul acestora este mai mare decât necesarul sarcinilor). Acest aspect poate duce la o altă problemă: datorită perioadelor de execuţie variabile (care pot fi relativ mari chiar şi pentru sarcini de mici dimensiuni), sarcina de mari dimensiuni poate fi ”înfometată” (nu va mai rula pentru că noi

sarcini mici sunt programate pe resursele aflate în aşteptare, care vor extinde – prin perioada de execuţie – posibilitatea ca noi sarcini mici să fie adăugate). Uzual, pentru a se evita această problema, se permite doar un număr limitat de sarcini mici care să fie prioritizate şi executate înaintea sarcinii mari, curente. Dacă acest număr este atins, procesoarele vor fi trecute în aşteptare, iar la eliberarea lor completa sarcina curentă va fi executată [9], [10].

Algoritmul EDF (Earliest Deadline First) se bazează pe un plan de execuţie, reprezentat sub forma unui tablou de planificări pentru fiecare din cele *m* maşini disponibile. Planificarea fiecărei maşini ia forma unei cozi de sarcini ce aşteaptă să fie rulate, ordonată după timpul de start al sarcini respective (în caz de concurenţă pe acelaşi moment de start, sarcina asignată unui procesor cu un ID mai mic este executată prima). În plus, se urmăreşte apariţia lacunelor în planificare – existenţa procesoarelor libere pe anumite perioade de timp (nu există sarcini care să necesite timp de procesor în acel moment) şi se menţine prezenţa acestora în coada asociată fiecărei resurse. Politica EDF adaugă fiecare sarcină nou venită (lucrează în mod dinamic) la planificarea iniţială. Nefiind necesară (tot timpul) recalcularea, această soluţie incrementală este mai rapidă şi mai economică computaţional decât declanşarea replanificării. Astfel se determină ce maşină (care respectă necesităţile programului) va rula sarcina sosită, care apoi va fi ataşată cozii maşinii respective [5].

C. *Simulatorul Alea 2*

Alea 2 (ajuns la versiunea 2.1) este un simulator bazat pe eveniment discrete, utilizat pentru testarea algoritmilor de planificare în diverse condiţii (resurse şi tipuri de sarcini diferite, modificări dinamice în mediul grid, etc.). Alea 2 conţine câteva module îmbunătăţite din aplicaţia GridSim – incluzând o politică de alocare a sarcinilor în grid, un mecanism de încărcare a maşinilor disponibile şi a sarcinilor de lucru (pentru formate de tip .gwf, .swf, Pisa .pwf sau MetaCentrum .mwf), un mecanism de încărcare a simulărilor de căderi de resurse, etc. Interacţiunea cu utilizatorul este de asemenea îmbunătăţita (faţă de GridSim sau de alte extensii precum GSSIM). Aceasta funcţionalitate este obţinută prin implementarea de clase-copil peste clasele părinte provenite din GridSim, ce permite extinderea relativ uşoară a simulatorului şi asigură compatibilitatea cu versiunile ulterioare de GridSim.

În cadrul simulatorului, maşinile aparţinând unui cluster pot fi co-alocate mai multor sarcini care rulează în paralel (atâta vreme cât există suficiente maşini libere), simultan, pe acelaşi cluster. Sarcinile care au un necesar de unităţi de procesare mai mare decât cel disponibil la nivelul clusterelor nu pot fi alocate către două clustere diferite şi vor fi respinse (o sarcină nu poate rula pe două sau mai multe clustere). De asemenea, dacă sarcina primită necesită anumite resurse speciale (un anumit sistem de operare sau o anumită aplicaţie instalată) care lipsesc din sistemul grid, respectiva sarcină este de asemenea respinsă. Pentru a permite aceasta, este necesar ca sarcinile primite (care reprezintă de obicei aplicaţiile client ce vor rula pe sistem – în cadrul acestui referat nu vom apela la sarcini interactive, care necesită intervenţia utilizatorului, acestea fiind mai mult prezente în gridurile comerciale orientate spre servicii) să sosească cu o serie de parametri specificaţi (inclusiv durata estimată de execuţie, termenul limită al execuţiei, etc.) care trebuie să fie luaţi în considerare de planificator atât pentru a stabili dacă sarcina poate fi rezolvată pe una din resursele aflate la dispoziţie (în caz negativ aceasta este remisă utilizatorului) cât şi pentru a stabili parametrii de asigurare a calităţii serviciilor care vor fi urmaţi. Aceştia pot fi aleşi din punctul de vedere al furnizorului (pentru a determina performanţa sistemului) sau din cel al utilizatorului (pentru a determina timpii de execuţie ai sarcinii furnizate) [11].

III. Algoritm îmbunătăţit pentru planificarea sarcinilor în grid

Dacă dorim să implementăm o metodă de planificare a sarcinilor nou sosite în grid folosind constrângeri QoS, este necesară o abordare care să ia în considerare rezervarea în avans a resurselor. Aşa cum s-a arătat anterior, în lipsa acesteia nu pot exista garanţii reale care să susţină un nivel ridicat de calitate a serviciilor.

În continuare va fi detaliat un algoritm de planificare bazat pe planuri de execuţie care va accepta constrângeri de calitate a serviciilor, care să distribuie cât mai uniform sarcinile în cadrul unui grid eterogen (a cărui resurse au un număr diferit de unităţi de procesare). În acest caz vom defini ca şi parametri de asigurare a calităţii serviciilor raportul dintre numărul de sarcini terminate cu succes (în limita termenului limită de execuţie propus) şi numărul total de sarcini (valide) primit. Un alt parametru important este responsivitatea sistemului; acesta poate fi definit atât obiectiv (din punct de vedere al sistemului) cât şi subiectiv (din punctul de vedere al utilizatorului). Deşi durata maximă de execuţie a setului de sarcini (*makespan*) este un parametru important care indică gradul de utilizare al resurselor (şi este de dorit maximizarea sa) şi responsivitatea sistemului este de asemeni importantă. Considerăm că o planificare optimă din acest punct de vedere nu înseamnă doar o distribuire a sarcinilor pe resursele cele mai puternice (cu cel mai mare număr de procesoare, până la epuizarea acestora) ci o distribuire uniformă, în care să se minimizeze numărul de resurse libere, care nu rulează nici un proces-client. Astfel, în algoritmii clasici de planificare, prezentaţi anterior, tendinţa este să se aloce procesele (după anumite criterii – de exemplu termenul limită de finalizare a execuţiei şi/sau durata estimată de execuţie) fie aleator, fie cu prioritate resurselor celor mai puternice (care conţin cele mai multe noduri de procesare). Acest lucru duce la o supra-utilizare a resurselor puternice şi la o sub-utilizare a resurselor mai slabe, sistemul devenind dezechilibrat dinamic din punct de vedere al sarcinilor rulate şi ducând la anumite dezavantaje: scăderea responsivităţii sistemului, creşterea congestiei resurselor, scăderea fiabilităţii platformei (o resursă supra-

aglomerată care iese din sistem afectează negativ planul de execuţie deoarece sunt mai multe sarcini care eşuează sau care trebuie replanificate).

La sosirea unei noi sarcini, planificatorul ordonează lista de resurse în funcţie de numărul de procesoare disponibile pe moment (fiecare resursă are planul său de execuţie precizând ce sarcini vor rula, când, şi pe câte unităţi de procesare) şi funcţionale (este posibil să apară erori la un moment dat în sistem, iar anumite noduri sau elemente de procesare din noduri să dispară ca urmare; lista trebuie actualizată dinamic). Se caută în lista de resurse cea cu cel mai mare număr de procesoare disponibile şi se testează dacă această resursă poate executa sarcina curentă. Dacă nu se poate executa (numărul disponibil de procesoare este prea mic), sarcina este trimisă primei resurse care va putea rula sarcina. Dacă se poate executa, se testează dacă plasând resursa pe respectiva poziţie noua planificare (candidatul) va avea o calitate mai bună decât cea anterioară (cu ajutorul funcţiei obiective). Dacă calitatea este mai bună, nou planificare va înlocui planificarea existentă. Căutarea poziţiei în care se va plasa noua sarcină (denumită gridlet) se face conform unui algoritm de tip backfilling, căutându-se o lacună potrivită în planul de execuţie global, care să îndeplinească cerinţele noi sarcini:

- să aibă un număr de procesoare egal sau mai mare decât cel cerut de sarcina în lucru;
- dacă sarcina specifică meta-date suplimentare (acesta este cazul când se foloseşte formatul .mwf, de exemplu, în contrast cu .swf sau .gwf), se testează dacă noua resursă răspunde proprietăţilor cerute de sarcină (se analizează politicile locale nodului comparativ cu cerinţele gridletului). Algoritmul este prezentat în continuare:

```
start
citeşte sarcina gi      // dinamic
creează dinamic lista de resurse disponibile (ri)
dacă ri != NUL atunci
   sortează lista descrescător (după nr. PE libere)
   pentru i=1,lungime_listă-1,i++ execută
      dacă !compatibilitate(ri,gi) atunci
         continuă parcurgere     // valoare i nouă
      altfel
         caută_slot_resursă_curentă(ri)    // tehnică backfilling
         dacă !slot(ri) atunci
            continuă parcurgere     // valoare i noua
         altfel
            foloseşte slot
            alocă gi planului de execuţie al ri
            criteriu_decizie_calitate(plan_execuţie(ri));
            dacă !calitate atunci
               elimină gi din plan de execuţie ri
            altfel
               actualizează ri
            sfârşit_dacă
         sfârşit_dacă
      sfârşit_dacă
   sfârşit_pentru
   actualizează resursele
sfârşit_dacă
calculează timp execuţie
stop
```

Funcţia obiectiv aleasă este cea care determină dacă – dat fiind o sarcină de planificat, un slot disponibil înaintea unor alte sarcini – sarcina poate fi plasată în respectivul slot cu creşterea (sau păstrarea) calităţii planului de execuţie. Dacă noua poziţie de plasare duce la îmbunătăţire, sarcina este acceptată şi planificarea actualizată. Pentru calcularea calităţii noului plan de execuţie, acesta se compară cu cel vechi. Simulatorul Alea 2 permite compararea după câteva funcţii obiective prestabilite. Dintre acestea menţionăm cea ce determină predictiv timpul mediu de încetinire (care măsoară responsivitatea sistemului şi este un factor important pentru furnizorul de servicii) - predictAvgSlowdown(current_time) - şi o funcţie ce determină predictiv numărul de sarcini care nu vor fi întârziate (aceasta măsoară nivelul calităţii serviciilor oferite către client) – predictNumberOfJobsThatMeetDeadline(current_time). Dacă noua variantă are un număr mai mare de sarcini ce nu vor fi întârziate, are loc planificarea noii sarcini şi actualizarea planului de execuţie.

IV. Date experimentale

Algoritmul de mai sus a fost implementat cu ajutorul simulatorului de sisteme grid Alea 2. Selecţia algoritmului a fost făcută din interiorul programului (el a fost introdus ca şi metodă printre celelalte metode ce conţineau algoritmi de planificare) şi au fost rulate teste comparative cu algoritmii FCFS şi EDF.

Seturile de date de intrare sunt preluate dintr-un sistem real, sub forma fişierelor de intrare. Acestea conţin şi o listă a resurselor disponibile, împreună cu caracteristicile acestora, ce modelează configuraţia gridului de pe care au fost preluate sarcinile. Fişierele de intrare folosite este preluat de la un sistemul Metacentrum (14 resurse cu 6, 8, 10, 16, 20, 32, 44, 64, 64, 70, 80, 92, 148, 152 procesoare) cu 50000 de sarcini testate (preluate sub forma de fişiere .mfw ce permit specificarea unor căderi de resurse pentru a se evidenţia comportamentul sistemului în condiţii realiste precum şi stabilirea unor resurse de tipuri diferite pentru nodurile sale).

Sunt testaţi trei algoritmi (FCFS, EDF şi algoritmul îmbunătăţit) şi se evidenţiază sporul de performanţă oferit de acesta din urmă faţă de cele etalon (Fig. 1, 2, 3). Fiecare algoritm a fost analizat prin prisma raportului dintre numărul de sarcini în aşteptare comparativ cu numărul de sarcini ce rulează (care poate oferi informaţii cu privire la lungimea perioadei de execuţie a unui set de date – *makespan* – care este de preferat să fie cât mai mic) şi numărul de procesoare folosit, comparativ cu numărul total de procesoare disponibil (care arată cât de eficient se folosesc resursele platformei) şi cu cel de procesoare cerut de setul de date de intrare (cu cât acesta

este mai mare, cu atât congestia resurselor este mai ridicată); o gestionare eficientă a sarcinilor din set poate preveni un număr ridicat de sarcini aflate în aşteptarea eliberării unor resurse de calcul (procesoare).

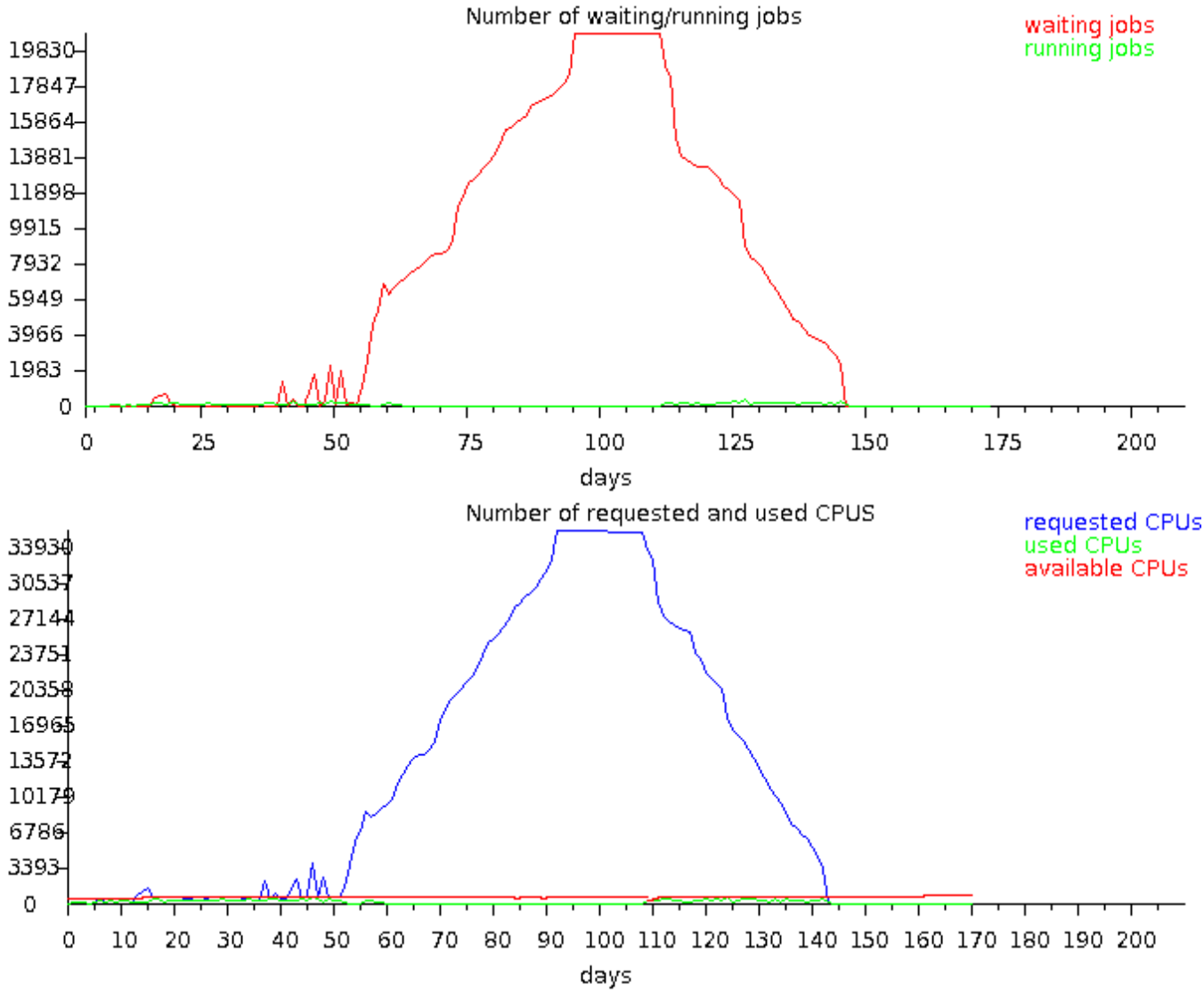
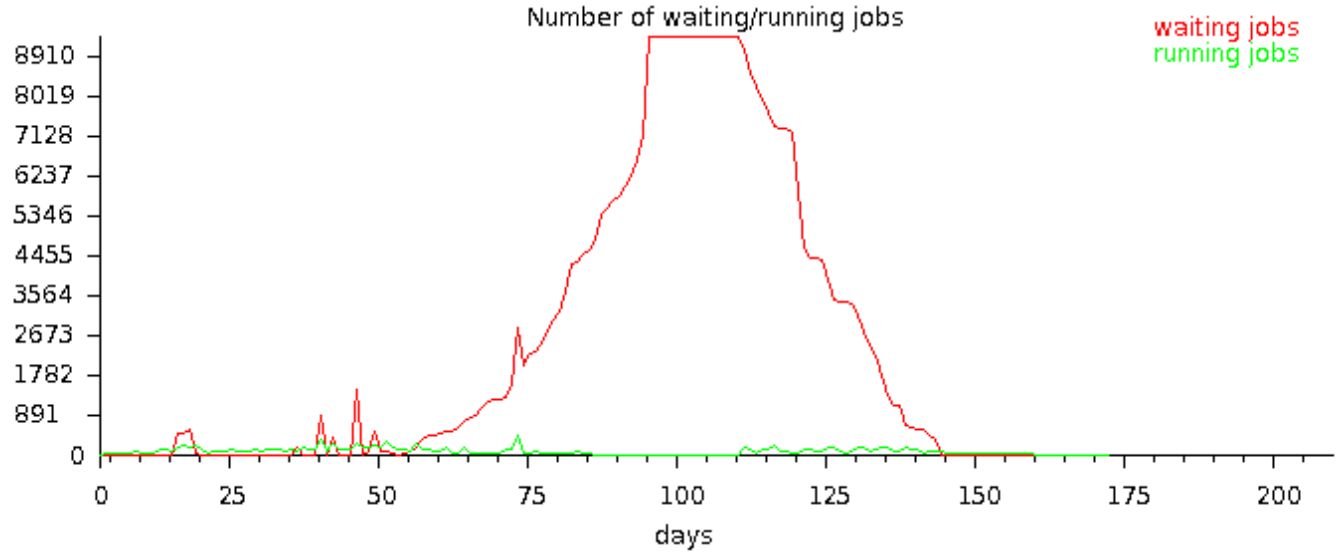


Fig. 1. Numărul de sarcini în lucru comparativ cu cel al sarcinilor în aşteptare şi utilizarea resurselor de calcul (procesoare) pentru algoritmul FCFS.



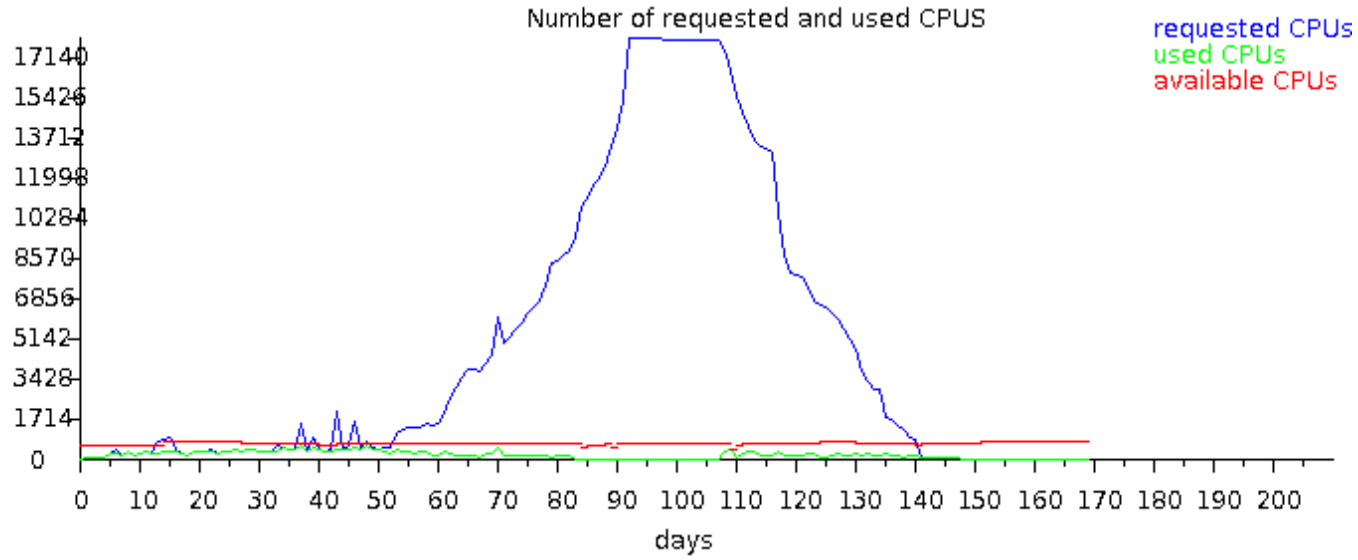


Fig. 2. Numărul de sarcini în lucru comparativ cu cel al sarcinilor în aşteptare şi utilizarea resurselor de calcul (procesoare) pentru algoritmul EDF.

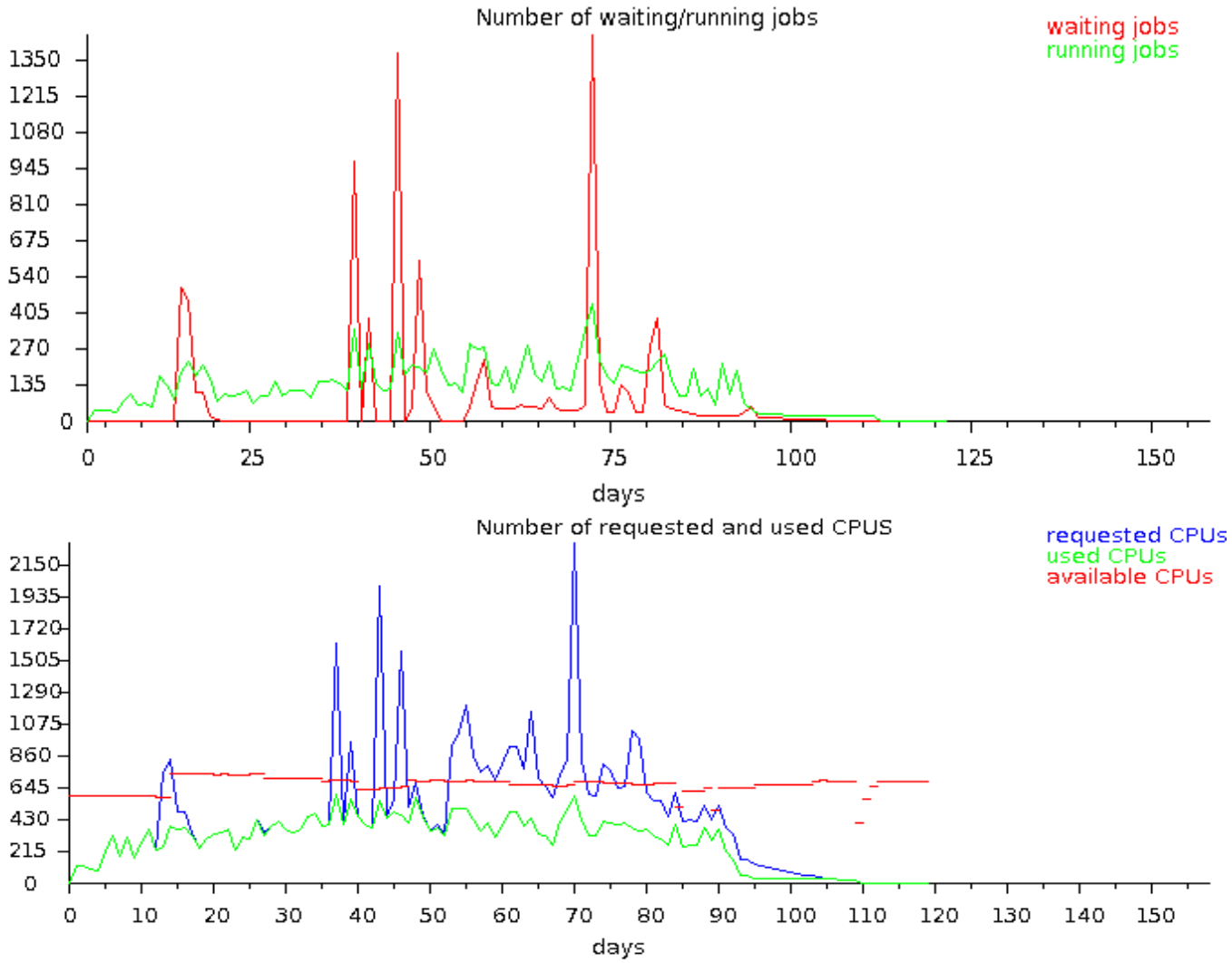


Fig. 3. Numărul de sarcini în lucru comparativ cu cel al sarcinilor în aşteptare şi utilizarea resurselor de calcul (procesoare) pentru algoritmul îmbunătăţit.

După cum se poate observa, algoritmul de planificare îmbunătăţit este mai rapid pentru un set de date de intrare dat (oferă un *makespan* redus), iar numărul de sarcini aflate în aşteptare (pentru eliberarea de resurse sau utilizarea programării) este mult mai scăzut (acest lucru se traduce printr-o distribuţie mai eficientă a sarcinilor – mai multe

sarcini rulate cu succes pe o perioadă determinată de timp – lucru benefic atât pentru utilizator cât şi pentru furnizorul de resurse). Numărul de resurse folosite este variabil în timp deoarece o parte din acestea pot eşua la un moment dat (până sunt repornite) iar setul de date de intrare furnizează şi această informaţie planificatorului.

V. CONCLUZII

Planificarea execuţiei sarcinilor are un impact puternic în ceea ce priveşte eficienţa utilizării resurselor sistemelor de tip grid şi a gradului de utilizare a acestora. Datorită numărului mare de resurse eterogene a unui astfel de sistem şi a mediului dinamic, algoritmii adaptivi de planificare oferă o performanţă mai bună (sub aspectul optimizării utilizării de resurse şi a scăderii perioadei de execuţie a unui set dat de sarcini).

Algoritmul suportă şi îmbunătăţiri viitoare sub forma unor parametri suplimentari care pot fi definiţi ca şi funcţii obiective pentru determinarea calităţii planificării. Astfel, se poate folosi un rating (din partea utilizatorului şi/sau prin analiza activităţii istorice a sistemului) pentru stabilirea unui sistem de reputaţie care să constituie primul factor de selecţie a resursei pentru o nouă sarcină (sarcinile vor fi asignate – în funcţie de prioritate – resurselor cu reputaţia cea mai bună). Acest rating va fi aplicat resurselor (nu gridului ca atare).

RECUNOAŞTERE

Setul de date MetaCentrum a fost oferit de Czech National Grid Infrastructure MetaCentrum prin intermediul Grid Workloads Archive. Simulatorul Alea 2 a fost oferit de Dalibor Klusáček şi Hana Rudová [12].

REFERINŢE BIBLIOGRAFICE

[1] F. Dong, S. G. Aki, “Technical report 2006-504: Scheduling algorithms for grid computing: State of the art and open problems”, 2006, disponibil la http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.69.3660.

[2] N. Fujimoto, K. Hagihara, ”A comparison among grid scheduling algorithms for independent coarse-grained tasks”, in International Symposium on Applications and Independent Workshops (SAINT 2004), 2004.

[3] R. Buyya, M. Murshed, “GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing”, in Concurrency and Computation: Practice and Experience, no. 14, pp. 1175-1220, 2002.

[4] R. Buyya, D. Abramson, J. Giddy, “An economy driven resource management architecture for global computational power grids”, in Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas, SUA, June 2000.

[5] D. Klusáček, H. Rudová, “Improving QoS in computational grids through schedule-based approach”, in Scheduling and Planning Applications Workshop (SPARK) at the International Conference on Automated Planning and Scheduling (ICAPS'2008), Sydney, Australia, 2008.

[6] Í. Goiri, F. Julià, J. O. Fitó, M. Mancías, J. Guitart, “Resource-level QoS metric for CPU-based guarantees in cloud providers”, in 7th International Workshop on Economics of Grids, Clouds, Systems and Services (GECON 2010), ed. Springer Verlag, pp. 34-47, Ischia, Italy, 2010.

[7] A. Mandal, K. Kennedy, C. Koebel, G. Marin, J. Mellor-Crummey, B. Liu, L. Johnsson, ”Scheduling strategies for mapping application workflows onto the Grid”, in Proceedings of 14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14), 2005.

[8] X. He, X. Sun, G. Von Laszewski, “A QoS guided scheduling algorithm”, in International Workshop on Grid and Cooperative Computing (GCC02), 2002.

[9] D. G. Feitelson, A. M. Weil, ”Utilization and predictability in scheduling the IBM SP2 with Backfilling”, in 12th International Parallel Processing Symposium, Orlando, USA, 2003.

[10] D. Talby, D. G. Feitelson, ”Improving and stabilizing parallel computer performance using Adaptive Backfilling”, in Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), Papers, vol. 1, 2005.

[11] D. Klusáček, L. Matyska, H. Rudová, ”Alea – Grid scheduling simulation environment”, in Parallel Processing and Applied Mathematics, pp. 1029-1038, Gdansk, Poland, 2007.

[12] D. Klusáček, H. Rudová, “Alea 2 - Job scheduling simulator”, in proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010), ICST, 2010.