

TDK dolgozat

Tóth Ádám

Debreceni Egyetem
Informatikai Kar
DE IK Informatikai Rendszerek és Hálózatok Tanszék

Véges forrású klaszter hálózatok hatékonyság analízise

Készítette:
Tóth Ádám
mérnökinformatikus MSc
1. évfolyam

Témavezetők:
Dr. Bérczes Tamás
adjunktus
Dr. Sztrik János
egyetemi tanár

Debrecen, 2014

Tartalomjegyzék

Bevezető	3
1. Bevezetés a grid rendszerekbe	6
1.1. Számítási gridek háttere	6
1.1.1. Definíciók	6
1.1.2. Számítási gridek előnyei	9
1.1.3. Egyéb számítási modellek	10
1.2. Job ütemezési problémák	12
1.2.1. Ütemezés új karakterisztikái a gridnél	12
1.2.2. Grid ütemezési kategóriák	13
1.2.3. Ütemező elvek a Gridnél	14
1.3. Energiakezelési problémák – mostani irányok	15
2. Rendszermodellezés háttere	18
2.1. Rendszerhátér	18
2.1.1. Rendszermodellezés	18
2.1.2. Rangsorolási módszertan	20
2.2. Rendszerfelépítés	22
2.2.1. Rendszerparaméterek	22
2.3. Terheltség	23
3. Puffer séma alapú modellek	24
3.1. Separate Queue	24
3.2. Class Queue	27
3.3. Common Queue	30
3.4. Energiametrikák	33

4. Numerikus eredmények és számolások	35
4.1. Rendszerjellemzőkkel kapcsolatos eredmények	35
4.2. Energiafogyasztás	40
5. Összefoglalás	46
Köszönetnyilvánítás	48
Irodalomjegyzék	51

Bevezető

A nagy sebességű hálózatok és az erős számítógépek fejlődése a gyors hardver költségek csökkenésével együtt az elosztott rendszerekre íródott alkalmazások elterjedéséhez vezetett, melyek szolgáltatásokat biztosítanak. Napjainkban egyre nagyobb szerepet kapnak a számítástechnikai hálózatok heterogén elosztott erőforrásainak használata, mint például a számítógépes grid rendszereké, így ezek vizsgálata kiemelt fontosságú. A minél hatékonyabb erőforrás-kihasználás végett, a jobok ütemezése komoly kihívások elé állítja a rendszerüzemeltetőket. A hálózathoz érkező jobok elosztása jelentős befolyással rendelkezik mind a rendszerhatékonyságára, mind annak energiafelhasználására. Az ütemező algoritmusok csoportosíthatóak néhány különböző rendszerkövetelmény szerint, melyek közül a következők a legtöbbször vizsgáltak: QoS [3, 4, 5], energiafogyasztás illetve ehhez a témakörhöz csatlakozó különböző kiszámított költség [6, 7], terheléselosztás [8] stb. A grid méret és a felhasználószám rohamos növekedésének következtében az energiafogyasztás egy kiemelt figyelmet igénylő téma lett, melynek figyelembe vétele nélkülözhetetlen a grid rendszerek működésének vizsgálata esetén. Erre a dolgozatra kitűzött feladatok a következők voltak:

- Klaszter hálózatok leírására képes program segítségével olyan modellek implementálása, mellyel a különböző puffer sémák összehasonlíthatóak
- Különböző puffer sémájú rendszerek energiafogyasztásának vizsgálata
- Legalább egy DPM (Dynamic Power Management) módszer alkalmazása és a hatásának vizsgálata

Ebben a dolgozatban egy olyan általános heterogén erőforrásokból álló számítógépes modelleket vizsgáltam, amelyek között a különbség a puffer sémák alkalmazásában mutatkozik meg. A klasztereket alkotóelemek rangsorolási módszere a Standard Power Evaluation Corporation tesztelő rendszer által készített SPECpower_ssj2008-on alapszik [9]. Így a rendelkezésre álló számítógépeknek a csoportba sorolása két szempont szerint történt:

- Teljesítmény (Shortest Queue with High Performance priority)
- Energiafelhasználás (Shortest Queue with Energy Efficiency priority)

A hatékony ütemezésen kívül a számítógépes gridben az energiahatékonyság nagyon fontossá vált a grid méret növekedésének és a „zöld” hálózat elérésének köszönhetően az utóbbi években. Ilyen típusú rendszerek esetén a legjobban hasznosítható módszerek az energiafogyasztás csökkentésére a Dynamic Power Management (DPM), melyeket futásidőben alkalmaznak. DPM-mel lehetséges tetszőlegesen kiválasztott rendszer komponenseket kikapcsolni, amikor azok tétlenek vagy dinamikusan lejjebb venni a működési frekvenciájukat és feszültségüket, amikor kihasználatlanok. Számítási rendszereknél energiamegtakarítás probléma kezelés céljából a heterogenitás és a nagymértékű erőforrás megosztás egy új stratégiaként használható, így különböző erőforrások elérhetőek egy szolgáltatásbiztosító által birtokolt adminisztratív tartományban, mint például egy számítógépes klaszter esetében. Továbbá megvalósítható az, hogy a heterogén erőforrások profitálása céljából hasznosítani lehessen az új erőforrás csoportot a régi helyett a teljesítmény vagy energiahatékonyság növelése érdekében. Ezt a megközelítést Zikos és Karatza mutatta be [7]-ben, ahol három elvet, számítási szintű ütemezésre alkalmazva (intenzív jobok a klaszternél), javasoltak: SQEE (Shortest queue with energy efficiency priority), SQHP (Shortest queue with high performance priority), és PBP-SQ (Performance-based probabilistics – shortest queue). A szerzők arra a következtetésre jutottak, hogy a PBP-SQ teljesít a leggyengébben és átlagos energiafogyasztást szolgáltat, az SQEE a leghatékonyabb energiahatékonyság mellett a legkisebb teljesítményt nyújtja, míg a SQHP túlteljesíti a másik kettőt a legtöbb energiát felhasználva. A szakirodalomban tekintett hagyományos számítási rendszereknél a klaszter hálózatot alkotó számítási elemek modelljei általában úgy néznek ki, hogy minden egyes elemhez egy sor tartozik.

Természetesen a figyelembe vett alkotóelemek processzorai teljesítménykapacitású és energiafogyasztású paraméterekkel rendelkeznek, amelyeket a Standard Performance Evaluation Corporation határozott meg hosszabb tesztelés után. A rangsorolási módszer, melyet vagy a nagy teljesítményű vagy az energiafogyasztási prioritás határoz meg, elősegíti a rendszert annak szisztematikusabb működéséhez. A PBP-SQ rossz viselkedésének következtében csak az SQEE és az SQHP elvet alkalmaztam a dolgozatban. A jobok tárolásához három puffer sémát vizsgáltam: Seperate Queue, Common Queue és Class Queue. A kapott eredmények azt mutatják, hogy a Class Queue és Common Queue határozottan túlteljesíti a Seperate Queue-t anélkül, hogy az energiafogyasztásra hatással lenne, ezen felül a jobok várakozási idejének csökkenése

miatt a Class és Common Queue puffer sémák jobb teljesítményt nyújtanak a Seperate-hez képest.

A dolgozat további része a következőképpen van strukturálva: az 1. fejezetben a számítási grid és ütemezésének háttéréről adok ismertetőt, majd a 2. fejezetben leírom a dolgozat alapjául vizsgált általános modellt és fontosabb jellemzőit.

A 3. fejezetben bemutatom az általam vizsgált puffer sémákat és azok működésének algoritmusait.

A 4. fejezetben a modellekkel kapcsolatos eredményeket ismertetem, melyek a dolgozatban modellezett rendszerek fontosabb rendszerjellemzőivel és energiahatékonyságával kapcsolatos.

Végül az 5. fejezetben a dolgozatomban leírt fontosabb dolgokat foglalom össze.

1. fejezet

Bevezetés a grid rendszerekbe

A grid egy tipikus modell a nagy teljesítményű elosztott számítási rendszereknél. Ebben a fejezetben, egy áttekintést adok ezekről a rendszerekről és a hozzájuk kapcsolódó problémákról. A 1.1. részben a fontosabb definíciók és a grid rendszerek előnyei lesznek ecsetelve. A 1.2. részben a grid rendszerekkel kapcsolatos ütemezési problémák leírását tartalmazza. A 1.3. rész az energiakezelési technikák rövid ismertetőjét tartalmazza, melyek fontossá váltak a hatalmas és komplex rendszereknél, mint a grid esetén, energiamegtakarítás céljából.

1.1. Számítási gridek háttere

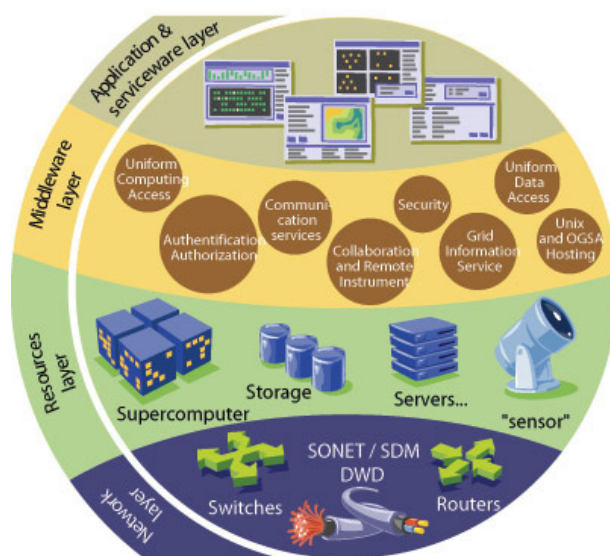
1.1.1. Definíciók

1990-es évek közepén egyetemekenél kezdett kibontakozni a számítási hálózat (computational grid) azzal a céllal, hogy megkönnyítse a felhasználók számára a más központokban lévő erőforrásokat kihasználni. Nagyon sok definíció született a grid rendszer kapcsán, de a legjobb definíció talán a [10]-ben található Three Point Checklist fejezi ki. Ebben Foster és a többiek így definiálták a gridet:

- egy rendszert, amely koordinálja azokat az erőforrásokat, amelynek alanyai nem központi módon menedzseltek,
- nyílt, általános célú szabványok szerint működnek és valamilyen kibővült minőségi szintet valósítanak meg

A grid rendszerek magját az úgynevezett köztes réteg (Grid middleware) adja, amely az operációs rendszer felett lévő szoftverréteg, de az alkalmazási réteg alatt helyezkedik el. Így ez a köztes

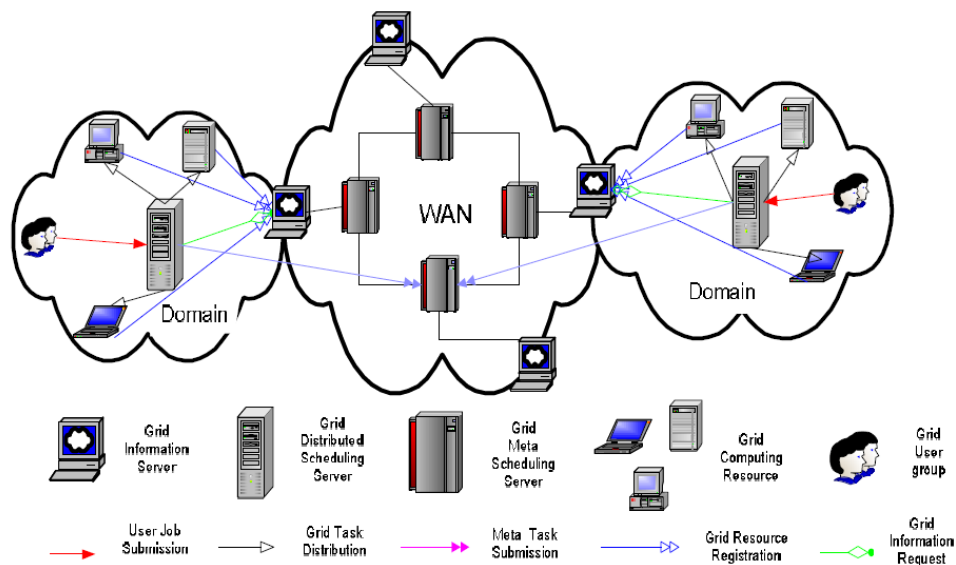
réteg biztosítja a közös programozói absztrakciót az elosztott rendszeren keresztül a számítógépek között a program részletek el- és szétosztásához. Ez egyben azt fejezi ki, hogy a számítás elosztott módú. A grid mérete terjedhet egy egyszerű laptoptól egészen heterogén hálózaton keresztül összeköttetésben lévő számítógépek és szuperszámítógépekből álló klaszterekig. A grid rendszert felépítő rétegeket az 1.1. ábra mutatja meg.



1.1. ábra. Grid rendszer rétegei [11]

Több típusú grid létezik, amelyeket a biztosított szolgáltatás különböztet meg egymástól. Ilyenek például a számítási, adat, alkalmazási és információs gridek. A számítási grid a fontos típusok egyike, amely számítási szolgáltatásokat biztosít. Ebbe bele tartozik az alkalmazási jobok végrehajtásához a biztonsági szolgáltatások nyújtása az elosztott számítási erőforrásokon egyetemesül vagy együttesen. Azt meg kell említeni, hogy ez a dolgozat csak számítási célra összehoztosít, így mostantól grid rendszer alatt a számítási gridet fogjuk érteni. A számítási grid elosztott modelljét a 1.2. ábra tartalmazza (lásd. [12]). A felső szint a Grid Information Servereket (GIS) és Grid Meta Scheduling Servereket (GMSS) tartalmazza. A második szint több tartományt foglal magába és minden tartomány Grid Distributed Scheduling Server (GDSS) tartalmaz. A harmadik szint többféle Grid Computing Resourcesból (GCR) és Grid User Groupsból (GUG) áll. Ezeknek a komponenseknek a funkcionalitása a grid modellben az alábbiak:

- *Grid Information Server (GIS)*: dinamikusan gyűjti az információt a regisztrált erőforrá-



1.2. ábra. Elosztott számítási Grid modell [12]

sokról és terjeszti az információt más GIS-eknek. GIS egy létfontosságú része akármilyen grides szoftver infrastruktúrának, és minden egyes tartományt legalább egy GIS irányít. A GIS kap grid információs requests-eket (kéresek) a GMSS-től, GDSS-től és GUG-tól, ezután megfelelő erőforrásokat keres, és visszaküldi az összes kielégítő erőforrásról szerzett információt a kérőknek.

- *Grid Meta Scheduling Server (GMSS)*: a különböző helyi ütemező ütemezésének összehangolására fókuszál, hogy ugyanannak az erőforrás halmaznak az elosztása ne történhesen meg különböző GUG-ra, melyek egyidejűleg igényelnek kéréseket. Minden GMSS elfogad meta ütemezésű kérést és együttműködnek más GMSS-ekkel a rendszer áteresztőképességének növelése érdekében és visszaküldi az információs kéréseket a GIS-nek.
- *Grid Distributed Scheduling Server (GDSS)*: Kulcsfontosságú elemek a grid felépítésében. Mikor a GUG átenged egy jobot a GDSS-nek a helyi tartományban, a GDSS kapcsolatba lép a GIS-sel, hogy a griddel kapcsolatos információkat gyűjtsön. Ezután a GDSS meta ütemezésű kérést küld a GMSS-nek úgy, hogy felhasználja a döntést hozó modulját a kérések ütemezésének és elküldésének érdekében. Végül a feladatokat a kiválasztott GCR-ek felé továbbítja.
- *Grid Computing Resources (GCR)*: Lehetnek munkaállomások vagy személyi számítógépek, melyek lehetnek homogének vagy heterogének. Minden GIS-hez regisztrált GCR a

GDSS és GMSS által végzett feladatok összerendelésének, feltérképezésének célpontjai.

- **Grid User Group (GUG):** A GCR-ek használatát igényli. A GUG kölcsönösen kommunikál a Grid környezetével a GDSS-en keresztül.

1.1.2. Számítási gridek előnyei

Ebben a részben a számítási gridek néhány fontos előnye lesz felsorolva a szükségességük megértéséhez:

- *Hatékony erőforrás felhasználás:* Egy létező alkalmazás futtatása a gridben máshol lévő (különböző) gépen a számítási grid alapvető felhasználása. Ekkor az a gép, amelyiken az alkalmazás normálisan fut, szokatlanul leterhelt állapotban lehet a tevékenységének csúcspontja miatt. Ez teszi lehetővé a hatékony kiaknázását a ki nem használt erőforrásoknak. A legtöbb asztali számítógép az időnek kevesebb, mint az 5%-ában van foglalt állapotban egy munkanapon keresztül. Néhány szervezetnél még a szerver gépek is gyakran lehetnek relatíve tétlen állapotban. A számítási grid biztosít egy keretrendszert a ki nem használt erőforrások kiaknázáshoz és így lényegesen növeli az erőforrások hatékonyságának kezelését. Következésképpen jobban kiegyensúlyozza az erőforrások kihasználtságát a különböző szervezetek között.
- *Párhuzamos számítás:* Ez egy szemmel láthatólag potenciális és vonzó tulajdonság a grid-nél. Ez azt jelenti, hogy az alkalmazás kisebb rész jobokra osztható és futtatható különböző erőforrásokon a griden keresztül.
- *Virtuális erőforrások és virtuális szervezetek az együttműködésért:* A számítási grid támogat egy környezetet az együttműködéshez a minél szélesebb közönség számára. A grid felhasználói dinamikusan beszervezhetőek virtuális szervezetekbe és ezek mindegyike különböző elvi (policy) követelményekkel rendelkezhet. Ezek a virtuális szervezetek megoszthatják erőforrásaikat az együttes használat végett.
- *Megbízhatóság:* A megbízhatóságú szemlélet a grid technológiában mind hardverre és szoftverre is vonatkozik, ellentétben a hagyományos rendszerekkel, ahol drága, feladat-orientált hardvert kell hozzávenni a hiba észleléshez és annak kezeléséhez. Grid menedzsment szoftver automatikusan átengedheti a jobokat hiba detektáláskor más gépekhez. Néhány kritikus, valós idejű szituációban a fontos joboknak több másolata lehet, amelyek

különböző gépeken futhatnak a griden keresztül és így az eredményeik felhasználhatóak inkonzisztencia ellenőrzésre, ilyenek például a számítógép hibák, adat korruptió és így tovább.

1.1.3. Egyéb számítási modellek

Számítási grid mellett, két másik nagy teljesítményű számítási modell létezik beleértve a számítógép klasztert és a számítási felhőt.

Számítógép klaszter

Számítógép klaszter tekinthető a számítási grid alrendszereként kisebb mérettel és költséggel. A klaszter párhuzamos vagy elosztott számítógépek kollektívja, amelyek nagy sebességű hálózaton keresztül kapcsolódnak egymáshoz. Ezek a számítógépek együtt dolgoznak, hogy olyan számításigényes vagy adatintenzív feladatokat hajtsanak végre, amelyeknek a kivitelezése egy számítógéppel nem valósítható meg. Számítógép klasztert főleg a magas rendelkezésre állás miatt használnak, mivel redundáns erőforrásokat tart fenn, amelyeknek használatára akkor van szükség, mikor bizonyos rendszerelemek meghibásodnak. Így terheléelosztást használva számítási munkaterhelést oszthatnak meg egymás között az alkalmazás kiszámolásához.

A klaszterek hardveres konfigurációja lényegesen változhat a választott hálózati technológiától és a rendszer elé kitűzött céltól függően. A fürtök kialakítása tipikusan három sajátosságból ered: az úgynevezett elosztott lemez, tükrözött lemez, és a nem megosztott konfigurációkból. Az elosztott lemezű klaszterek a lemez elérésének végett a közös I/O buszra támaszkodnak, így az I/O eszköz elérhető a klaszteren belül minden számítógép számára. Mivel minden csomópont (számítógép) egyidejűleg a központi lemezekről írhat vagy tárolhat adatot, így egy szinkronizációs mechanizmus az úgynevezett Distributed Lock Manager (DLM) használata szükséges, hogy a rendszer konzisztenciáját fenntartsa. Elosztott lemezű klaszterek magas rendelkezésre állást nyújtanak mivel egy csomópont meghibásodása esetén más csomópontokra nem lesz kihatással. A nem megosztott klaszterek esetén több csomópont egyidejűleg nem férhet hozzá a lemezhez így nincs szükség Distributed Lock Manager használatára. Tükrözött lemezű klaszteres megközelítésnél a tükrözés magába foglalja az összes alkalmazás adatának többszörösítését a fő tárolóból a másodlagos tartalékra, esetleg egy távoli helyen lévőre elérhetőségi szándékból.

Számítási felhő

Ez a modell 2007 végén jött létre arra az alapelve építve, hogy a helyi számítógép által végzett számítások átvihetők a hálózatba. A felhőt úgy lehet definiálni, mint olyan párhuzamos és elosztott rendszer típus, mely egymással összekötött, virtualizált számítógépeket tartalmaz [13]. Gondoskodik az újrakonfigurálható számítási erőforrások osztott tartományairól, amelyeket a felhasználók elérhetnek az interneten keresztül. Három féle típusú szolgáltatásbiztosítást különböztetnek meg a felhő számítási modelleknél:

- Software as a Service (SaaS), ahol sok felhasználó használatba veheti a szolgáltatásbiztosító által támogatott szoftvert és csak a felhasznált időnek megfelelően fizet.
- Platform as a Service (PaaS) magas szintű integrált környezet, melyet lehet tervezni, felépíteni, tesztelni, telepíteni és így tovább.
- Infrastructure as a Service (IaaS) a számítógépes infrastruktúra bérletét jelenti, beleértve szervereket, az ahhoz kapcsolódó szoftvereket és a hálózati eszközöket is. Így olyan szolgáltatásokra utal, melyek a felhasználók számára biztosítottak és használhatják teljesítmény, tárolás, hálózat és egyéb számítási erőforrások feldolgozására, hogy bármilyen szoftvert futtathassanak beleértve operációs rendszereket és alkalmazásokat. A felhasználó azt használ, amire szüksége van az interneten és csak azok után fizet, amit használ.

Számítási felhő vs számítógépes klaszter

Technológiai szempontból nézve a számítási felhő nem egy új technológia, de inkább tekinthető a számítógépes klaszter viszonylagos fejlődésének. Így ennek a két modellnek vannak hasonlóságai, mivel mindkettő dinamikus erőforrás halmazok virtualizációs technológiájára támaszkodik és szolgáltatás szintű megegyezést használ a szolgáltatásbiztosítás menedzselésére. Mindkettő nagyban skálázható, és mindkettő lehetővé teszi több ügyfél kiszolgálását (multi-tenancy). Ezenfelül támogatják a multitaszkingot, ami azt jelenti, hogy több felhasználó végrehajthat különböző feladatokat és hozzáférhet egy vagy több alkalmazás példányaihoz. Viszont néhány témában észrevehetően különbözik a számítási felhő és a számítás grid. Ezek közé tartozik az üzleti modell, erőforrás-kezelés, erőforrást biztosító modellek, és az erőforrások elérhetősége [14]. A különbségeket az alábbiak írják le:

- Grid üzleti modellek egyetemi intézmények között kétoldalú megállapodás alapúak, míg a felhők jobban szétválasztott üzleti modelleket követelnek meg erőforrás szolgáltatók-

tól (akik csak számítási erőforrásokat biztosítanak) egészen a SaaS szolgáltatókig (akik együtt adnak el erőforrásokat a szoftveres szolgáltatásaival).

- Az erőforrás-kezelési megoldást tekintve, gridek kötegelt rendszereken alapszanak, míg a felhők virtualizációs technológiákat használnak.
- Erőforrás-biztosításban nyilvánvalóan különbözik a felhő és a számítási grid. Grideknél a kapcsolat kiépítés egyetemi intézmények között offline módon történik, míg felhőknél igény szerinti erőforrás-biztosítás van, ami kizárja a grideknél létező túlbiztosítást (over provisioning).
- A számolási felhő sokkal rugalmasabban szolgáltatja az erőforrásokat a gridekhez képest, ahol az erőforrás-megosztás „best effort” jellegű, így néha nincs elég elérhető erőforrás és néha rengeteg tétlen erőforrás áll rendelkezésre.
- Adattárolási kapacitás egy másik jelentős különbség, ezt illetően a felhők előnyösen teszi lehetővé az objektumok tárolását, melyek mérete egészen bájtoktól gigabájtokig változhat vagy akár terabájtosak is lehetnek, míg a gridek nagyon alkalmasak adatintenzív tárolásra és az adatnak elég nagyoknak kell lennie a maximum haszon, előny eléréséhez.

1.2. Job ütemezési problémák

Ahogy a feladat ütemezés bármely számítási rendszer esetében lényeges és fontos probléma, úgy az új rendszer karakterisztikáknak köszönhetően a gridnél egyre kihívásosabbak a mechanizmusok. Ebben a részben a gridek új, fontos karakterisztikája lesz bemutatva, melyeket az ütemezésnél figyelembe kell venni. Ilyen például a különböző ütemezési szintek specializációja. Ezen felül néhány alapvető ütemezési elv lesz megemlítve, melyek tanulmányozása mostanában egyre nagyobb jelentőséggel bír.

1.2.1. Ütemezés új karakterisztikái a gridnél

Ütemezési probléma a legjobban tanulmányozott problémák egyike az optimalizáló kutatóközösség számára. Azonban van néhány jellemző, ami a problémát mássá, kihívásosabbá teszi a grid rendszerek esetén:

- *Dinamikus struktúra:* A hagyományos elosztott rendszerekkel ellentétben a grid rendszernél az erőforrások megjósolhatatlan módon csatlakozhatnak vagy távozhatnak. Így az

erőforrások átmehetnek különböző adminisztratív tartományokba, ami szimplán köszönhető a rendszerrel való kapcsolat elvesztésnek vagy, mert a tulajdonosok kikapcsolják a gépet vagy operációs rendszert váltanak stb.

- *Heterogenitás*: Grid rendszereknél a számítási erőforrások nagyon eltérőek lehetnek a számítási kapacitásukat tekintve, mely lehet laptop, asztali gép, klaszter, szuperszámítógép vagy akár kis számítási eszköz is. A jelenlegi grid infrastruktúrák nem mondhatók sokoldalúnak, de bármely grid rendszer esetében a heterogenitás az egyik legfontosabb tulajdonságok közé tartozik. Az erőforrások heterogenitása mellett bármely grid rendszert nézve a beérkező jobok a számítási igényük szempontjából különbözőek és heterogének lehetnek, mint például számításigényesek vagy adatintenzívek illetve a jobok lehetnek teljes alkalmazások vagy csak atomi feladatok. Fontos kiemelni, hogy általában a grid rendszernek nincs tudomása a rendszerbe érkező feladatok típusáról.
- *Nagy skálázás*: A grid rendszerektől elvárás, hogy nagyban skálázhatóak legyenek. Egy idő után a gridhez bocsátott alkalmazások, jobok vagy feladatok száma akár sok is lehet. Így a hatékony erőforrás-kezelés és jobok tervezése miatt szükség van különböző típusú ütemezők használatára (super-schedulers, meta-schedulers, decentralized schedulers, helyi ütemezőket, resource brokers stb..) és ezeknek a lehetséges hierarchikus kombinációjukra a skálázás eléréséhez.

1.2.2. Grid ütemezési kategóriák

Grid ütemezők olyan szoftverkomponensek, melyek a feladatoknak a gridben lévő erőforrásokhoz való összerendeléséért felelősek néhány kritérium és grides környezet konfiguráció mellett. A grid ütemezőn belül különböző szintek azonosíthatók tartalmazva super-schedulerseket, meta-schedulerseket, helyi/klaszter ütemezőket és vállalati ütemezőket. Ezekről a típusokról a következőekben lesz szó.

- *Super-scheduler*: Ez a típusú ütemező megfeleltethető a központosított ütemező megközelítéshez, amelynél helyi ütemezőket használták az erőforrások fenntartásához és elosztásához a gridben. A helyi ütemezők kezelik a jobokat sor feldolgozásnál. A super-scheduler felelős az előzetes erőforrás fenntartás, egyezkedés és szolgáltatás szintű meg egyezés kezelésért.
- *Meta-scheduler*: Ennek az ütemező fajtának a (meta-brokerként is ismert) szerepe akkor

adódik, amikor különböző rendszereken keresztül kell egy jobot vagy alkalmazást egynél több erőforráshoz elosztani. Mint a super-schedulerek esetén a meta-scheduler bizonyos rendszereknél helyi ütemezőket is használ. Így a meta-schedulerek koordinálni tudják a teljes ütemezés számításához a helyieket. A rendszerek közti terheléselosztás véghezvitele a meta-scheduler típusú ütemezők fő célja.

- *Helyi/klaszter ütemező*: Ez a típus felelős ugyanazon helyi területű hálózatban a feladatok kiosztásáért az erőforrásokhoz. Az ütemező kezeli a helyi erőforrásokat és a helyi sorbanállási rendszert.
- *Vállalati ütemező*: Ennek az ütemező fajtának szerepe a nagy cégeknél adódik, amikor számítási erőforrások szétosztása történik több vállalati osztály között. A vállalati ütemező ugyanahhoz a vállalathoz tartozó különböző helyi ütemezőket használ.

1.2.3. Ütemező elvek a Gridnél

Az előző részben tárgyalt ütemező kategóriák szerint az ütemező algoritmusok implementációja két szintű osztályozás szerint történhet: grid és klaszter szintű. Grid szintnél az ütemező algoritmusoknak azzal kell megbirkózni, hogy hogyan osszák szét a jobokat különböző helyi tartományokhoz, és ezután a helyi ütemező klaszter szinten a jobokat kiosztja a számításigényes erőforrásokhoz. A szakirodalmakban előforduló ütemezési elvek:

1. *Grid szintű elvek*:

- (a) *Minimum Jobs in Queue (MJQ)*: A GS-hez érkező jobok ahhoz a klaszterhez lesznek irányítva, amelynek processzor sorában összességében a legkevesebb job várakozik. Minden egyes klaszter esetén ki kell számítani a sorban várakozó jobok teljes számát és az a klaszter lesz kiválasztva, amelynél ez a szám a legkisebb.
- (b) *Less Power Consumed (LPC)*: Minden egyes klaszter esetén ki kell számolni a processzorok aktuális teljes energiafogyasztását. A GS-hez érkező jobok ahhoz a klaszterhez lesznek irányítva, amelynek az adott időben a legkevesebb az energiafogyasztása.
- (c) *Minimum Service Demand (MSD)*: Minden job érkezés (a klaszterbe) alkalmával a klaszterben lévő összes job esetén ki kell számítani egyenként az általuk igényelt

teljes kiszolgálási időt. A GS-hez érkező jobok ahhoz a klaszterhez lesznek irányítva, melynek a legkevesebb teljes kiszolgálási időigénye van a processzor sorában várakozó jobok közül.

2. *Klaszter szintű elvek:*

- (a) *Join the Shortest Queue (JSQ)*: Helyi ütemező (LS) a beérkező jobokat ahhoz a processzorhoz irányítja, melynek sorában a legkevesebb számú job várakozik.
- (b) *Minimum Service Demand (MSD)*: LS-hez érkező jobokat ahhoz a processzorhoz irányítja, melynek sorában a legkevesebb teljes kiszolgálási időigényű job van.

1.3. Energiakezelési problémák – mostani irányok

Manapság a technológiák és a számítógépes hálózatok méretének gyorsütemű fejlődésének köszönhetően az energiahatékonyság kritikus kíváncsálommá vált és érdekes tanulmányozási irányt nyitott meg az ütemezési stratégiák esetén. Ez az alfejezet egy rövid áttekintést nyújt az energiakezelési technikákról, amelyek célja, hogy a rendszerek minnél kevesebb energiát fogyasszanak. Elemzések részletesebben erről a témáról a [15, 16, 17]-ban találhatóak. Energiacsökkentési módszerek lehetnek statikusak vagy dinamikusak. Statikus energiakezelés (SPM) módszereket, mint például a kis energia módra való fordítás és szintézis, alkalmaznak tervezési időben (offline), ami mind a hardvert és szoftvert is érinti. Ezzel ellentétben Dynamic Power Management módszereket futásidő közben alkalmaznak (online) energiacsökkentés végett, amikor a rendszerek kis munkaterhelésű feladatokat szolgálnak ki, vagy amikor tétlenek.

Static Power Management (SPM) módszerek két területre oszthatóak: CPU és rendszerszintre. A CPU energiafogyasztása drasztikusan megnőtt és a vizsgálatok az egyik fő témájává nőtte ki magát. A szerkezeti fejlődés (egy szálról több szárra és egy magról több magos CPU) az elmúlt években a CPU energiafogyasztását sokkal hatékonyabbá tette. Rendszerszinten SPM módszerek alkalmazhatóak megcélózva hardvereket és szoftvereket különböző szintű absztrakciós szinten, mint például applikációs, operációs rendszer vagy teljes rendszerszinten. Teljes rendszerszint azt jelenti, hogy a CPU-t, memória hierarchiát, kis energiájú lemez alrendszereket modellezi, és mennyiségileg meghatározza az applikációs és operációs szinten használt energiát.

DPM felöleli azokat a módszereket, amelyek úgy érik el az energiahatékony számítást, hogy a rendszer komponenseket kiválasztva kapcsolja ki (vagy teljesítményét csökkenti), amikor azok tétlenek (vagy részben kihasználatlanok). Például a dynamic voltage and frequency scaling (DVFS) megváltoztatja a processzor feszültség ellátását vagy frekvenciájának működését futás-időben, ami egy fajta energiamenedzselés is egyben. DPM-t lehet arra is használni, hogy nem használt I/O eszközöket vagy ki nem használt klaszterbeli szerver csomópontokat lekapcsoljon. Mivel ezt a módszer típust a rendszer futásidejében alkalmazzák, így együtt tud működni ütemezési elvekkel az optimális ütemezési viselkedés elérésének céljából mind teljesítmény és energiahatékonyság szempontok szerint.

DVFS, ami egy CPU szintű módszer, egy nagyon népszerű eljárás, amelyet a modernkori processzorok támogatnak. Legyen P egy CMOS alapú processzor, amelyhez kapcsolódó órajelfrekvencia: f és működő feszültség: V , így a következő egyenlet az energiafogyasztás alapelvéből adódóan:

$$P = C \times V^2 \times f \quad (1.1)$$

Ahol C a kapacitás. Így a működési frekvencia és feszültség csökkentésével a CPU energiafogyasztása csökken. A nem valós idejű rendszereknél az alkalmazásoknak nincsenek számítási határideje, így a CPU használhatja a legkisebb frekvenciát a feladat végrehajtásához energiamegtakarítási szempontot figyelembe véve. Azonban a legtöbb alkalmazás rendelkezik explicit vagy implicit határidővel, ekképpen az alkalmazások határidejének kielégítésé DVFS alkalmazásnál egy megszorítás a valós idejű rendszereknél. A tipikus DVFS algoritmusok a teljesítményigények megjósolásához intervallum és feladat alapú ütemezés szerint kategorizálnak. Intervallum alapú algoritmusok az aktuális időintervallumot megelőző processzor használatát mérik és ezt az információt felhasználva az aktuális teljesítmény szintet állítja be a következő időintervallumra. Feladat alapú algoritmusok minden egyes feladathoz tartozó munkamennyiséget és határidőt elemzi és a processzor teljesítményét igazítja az eredmények alapján minden egyes feladathoz. Kereskedelmi termékeknél a szélesebb körben alkalmazott algoritmusok intervallum alapúak, mert egyszerűek és könnyű implementálni őket. Azonban gyakran hibásan jósolja meg a teljesítményigényt, ami nem várt végrehajtási késleltetést eredményezhet. Feladat alapú algoritmusok azt feltételezik, hogy minden feladatnak van vagy explicit, vagy implicit határideje. Mivel sok alkalmazásnak nincs időszerűségű követelményei, így ez a fajta feltételezés a

beágyazott multimédiás rendszereknél értelmetlen határidő detektálási többletterhelést okozhat.

Rendszerszintű DPM szempontnál az applikációs szintű DPM módszereknél az OS, CPU együttműködhet energiahatékonyság elérése érdekében. A tétlen számítási elemek be/ki kapcsolása közkedvelten használt valós idejű rendszereknél, mert a legtöbb rendszer jelentős időt tölt tétlen állapotban. Ennek a módszernek az előnyei közé tartozik, hogy annyi energiát tud megtakarítani amennyit csak lehetséges, és különböző szinteken alkalmazható (szerver/számítógép, CPU, mag). De meg vannak a maga hátrányai is: időt vesz igénybe a ki/be állapotba való átmenet és nem olyan egyszerű meghatározni, hogy mi az optimális idő/alkalom a rendszerben nem használt komponensnek kikapcsolásának. Ennek a módszernek az osztályozása az alábbi elvek alapján szokott történni.

- *Időtúllépéses/statikus elvek:* Ennél a kategóriánál ezekről az elvekről azt feltételezzük, hogy miután egy eszköz tétlen egy bizonyos időtúllépési értéken keresztül, akkor legalább még az állapotváltás költségének kompenzálásához szükséges minimum ideig (break-even time) tétlen állapotban marad. A tétlen periódus hosszának minimuma felhasználható energiamegtakarításra.
- *Jövendőelő elvek:* Kezds előtt megjósolja az ideális periódus hosszt az időtúllépés kiküszöbölése érdekében. Ha egy tétlen periódust úgy jósol meg, hogy hosszabb, mint az állapotváltás költségének kompenzálásához szükséges minimum idő, akkor az eszköz egyből elalszik, miután tétlen állapotba kerül.
- *Sztokasztikus elvek:* a bejövő kéréseket és az eszköz energia állapotú változásait modellezi sztochasztikus folyamatoknak megfelelően, ilyenek például a Markov-folyamatok.

Tipikusan minden elvet hat kritérium szerint minősítenek: energia, leállások száma, leállítás precizitása, interaktív teljesítmény, memória és számítási követelmények. Nem lehet olyan elvet találni, ami minden kritérium szerint a legjobb minősítésű lenne. Az említett elvek, amelyeket az eszközközkezelő implementál szűrő drivernek is nevezik, melyet beillesztenek az operációs rendszer kernele és egy másik eszközközkezelő közé.

2. fejezet

Rendszermodellezés háttere

Ebben a fejezetben a tekintett heterogén erőforrásos általános számítási klaszter modelljének ismertetéséről lesz szó. A 2.1. rész a használt modell háttérének leírását tartalmazza. A dolgozat további részében használt konkrét erőforrások és paraméterei lesznek kifejtve a 2.2. részben. A 2.3. részben a kiszolgálással kapcsolatos paramétereikről lesz részletesebben szó.

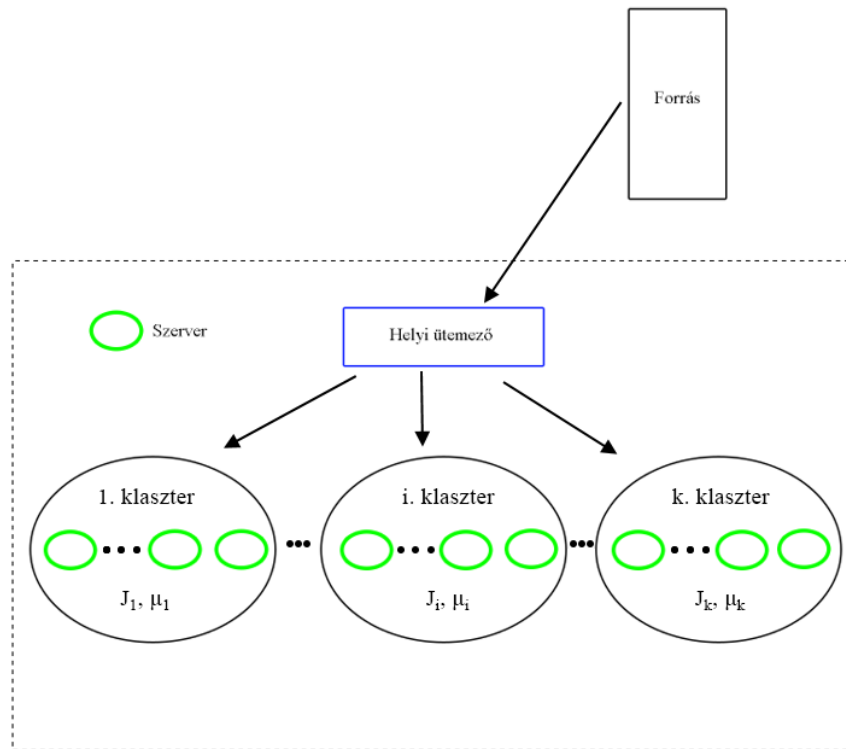
2.1. Rendszerhátér

2.1.1. Rendszermodellezés

A számítási klaszter összekötött számítógépek halmazát tartalmazza, amelyek egy célért dolgoznak együtt az úgynevezett „számítási küldetésért”. A klaszter komponensei általában egymással össze vannak kötve gyors helyi hálózatokon keresztül (LAN), ahol minden egyes csomópont (számítógépet szerverként használva) a saját operációs rendszer példányát futtatja. Ebben a dolgozatban tekintett klaszter heterogén, így a klaszterben lévő erőforrások különböző jellemzőkkel rendelkeznek. A szisztematikus job ütemezés implementálásához az ugyanazokkal a jellemzőkkel rendelkező szerverek egy csoportba sorolhatóak, így a helyi ütemezőnek nem kell megnéznie az összes szerver tulajdonságát, elég a szerverhez tartozó osztályét figyelembe vennie. A következő általános I szerver osztályból álló klasztert tekintjük, melyek indexelése $i=1, \dots, I$ -ig terjed ($I \geq 1$), lásd a 2.1. ábrát. Azt feltételezzük, hogy a tekintett klaszternek van helyi ütemezője, mely tudatában van a szerverek jellemzőivel és a rendszerállapotával és képes kiszolgálni a számításigényes jobokat, amelyekre a következők érvényesek:

- sorrendi (First Come First Served) kiszolgálási elv alkalmazása

- kiszolgálási igényük ismeretlen a helyi ütemező számára
- kiszolgálás alatt nem felfüggeszthetőek (nem előrelátó tulajdonság)
- atomi jobok, amelyek nem oszthatóak szét kisebb feladatokra
- bármely szerver feldolgozhatja



2.1. ábra. Általános heterogén klaszter modell

Azt feltételezzük, hogy a rendszerhez az igények véges forrásból érkeznek és minden forrásban lévő egyed generál jobot λ paraméterű Poisson-eloszlás alapján. Így a rendszerbe érkező jobok maximális intenzitása legfeljebb $N \times \lambda$ lehet, ahol N a forrásszámot jelöli. A kiszolgálási idő, ami alatt a szerver végrehajtja a jobot exponenciális eloszlású, így azt feltételezzük, hogy minden egyes szerver az i -edik osztályban μ_i kiszolgálási rátával rendelkezik és az i -edik osztály J_i processzort tartalmaz ($J_i \geq 1$). Az egész rendszer kiszolgálási rátája a következőként számolható ki:

$$\mu_{\text{system}} = \sum_{i=1}^I J_i \times \mu_i \quad (2.1)$$

A rendszerterheltség, a rendszer által szállított nagy mennyiségű forgalom, amely a beérkezési és kiszolgálási intenzitás közti arányaként definiálható:

$$\rho_{\text{system}} = \frac{\lambda \times \overline{N}}{\mu_{\text{system}}} \quad (2.2)$$

Meg kell jegyezni, hogy a tekintett modell véges forrású, így mindig létezik a stacionárius eloszlása, ami azt jelenti, hogy a rendszer mindig stabil.

2.1.2. Rangsorolási módszertan

Az általunk vizsgált modelleknél a teljesítmény és energiatudatos ütemezési elveket: SQEE-t és SQHP-t alkalmaztuk, melyet Zikos és Karatza vizsgált [7]-ban. Ezek az elvek röviden így jellemezhetőek:

- *SQEE (Shortest Queue with Energy Efficiency priority)*: Ez az elv a rendszerben lévő legrövidebb sorral rendelkező szervert választja. Ha több ilyen szerver van, akkor a legkisebb energiafelhasználásút választja. SQEE úgy szemléltethető, hogy a nagy teljesítmény árán a legnagyobb energiatakarékosságot nyújtja.
- *SQHP (Shortest Queue with High Performance priority)*: Hasonlóan az SQEE-hez, az SQHP először megtalálja legrövidebb sorral rendelkező szervert. Ha több ilyen szerver is van, akkor a nagyobb teljesítményű az elsődleges. SQHP-t úgy lehet szemléltetni, hogy túlteljesíti a többi elvet, de egyben több energiát is fogyaszt.

Számítási klaszternél minden egyes fizikai szerver egy meghatározott szerver típushoz tartozik. Jelölje S a szervertípusok halmazát és $I=|S|$ az S mérete. Tekintsük az s szerver típust, $s \in S$, amely a C_s , $P_{ac,s}$ és $P_{id,s}$ paraméterekkel jellemezhető ahol:

- C_s az egy ssj_ops érték, ami a következőként van definiálva: befejezett műveletek száma a mérés intervalluma alatt osztva az erre definiált intervallum másodperceinek számával. Ez a paraméter jellemzi a számítási szerver áteresztőképességét.
- $P_{ac,s}$ szerver fogyasztása terhelt állapotban a Standard Performance Evolution Corporation (SPEC) tesztelő rendszer alapján mért fogyasztás, ami a SPECpower_ssjs2008-ban található meg.
- $P_{id,s}$ a szerver energiafogyasztása, tétlen állapotban (amikor nincs kikapcsolva).

Azt feltételezzük, hogy amikor egy szerver foglalt, akkor C_s átmenő teljesítménnyel működik megközelítőleg $P_{ac,s}$ energiafogyasztással. Amikor a szerver tétlen, akkor ki lehet kapcsolni anélkül, hogy fogyasztana, így megtakarítva energiát, vagy szoftveresen meg lehet állítani a CPU fő belső óráját és így $P_{id,s}$ mennyiségű energiát fogyaszt. Érdekes még megemlíteni, hogy a SPEC mérési szabvány szerint, $C_s/P_{ac,s}$, $s \in S$, s osztály teljesítmény energia aránya, amely leírja a kiszámított alkotóelem energiahatékonyságát, úgy hogy a nagyobb arány nagyobb energiahatékonyságot jelent.

Két rangsoroló függvényt definiáltunk, amely energiahatékonyságon vagy nagy teljesítményen alapszik, amelyeket a következők írnak le:

Nagy teljesítmény esetén:

$$r_p(s) = \frac{C_s}{\max_{i \in S} C_i}, \quad s \in S \quad (2.3)$$

Energiahatékonyság esetén:

$$r_e(s) = \frac{\frac{C_s}{P_{ac,s}}}{\max_{i \in S} \frac{C_i}{P_{ac,i}}}, \quad s \in S \quad (2.4)$$

Jelölje S_p és S_e a rendezett szerver halmaz típusát. S_p sorrendnél (the high performance priority ranking) és S_e -nél (the energy efficiency priority ranking) a következő állításokat lehet kijelenteni:

- szerver típus beosztás 1-től $|S|=I$ -ig terjed a kiszámított $r_p(s)$ és $r_e(s)$ értékekből kiindulva, $s \in S$
- az első számú, a következő módon rendelődik a szerver típushoz: $\arg \max_s r_p(s)$ és $\arg \max_s r_e(s)$
- I -edik minősítésű, a következő módon rendelődik a szerver típushoz: $\arg \min_s r_p(s)$ és $\arg \min_s r_e(s)$

S_p minősítésnél, ha két olyan szerver típusú van ugyanazzal az ssj_ops értékkel, akkor a nagyobb átlagos aktív energiájú szerver típus kap nagyobb indexet.

Az ugyanolyan típusú fizikai szerverek alkotnak egy osztályt, melynél a sorrend vagy S_p (amikor a nagy teljesítményű prioritás lett választva) vagy S_e minősítés (amikor energiahatékonyság a preferált) szerint történik. Ez azt jelenti, hogy kisebb indexhez társított szervereknek van nagyobb prioritása.

2.2. Rendszerfelépítés

2.2.1. Rendszerparaméterek

A modellezésből kiindulva egy olyan klasztert építettem fel, amely három típusú Commercial Off-The-Shelf (COTS) szerverből ($I=3$) épül fel és mindegyik klaszter három szervert tartalmaz ($J_i=J=3, \forall i=1, \dots, I$), melyeknek a paramétereit a SPEC tette közzé [9]. A 2.1. táblázat tartalmazza ezeknek a típusoknak az adatait.

Szerver típus	C_s	$P_{ac,*}$ (W)	$C_s/P_{ac,*}$	$P_{id,*}$ (W)
Acer AW2000h-Aw170h F2 (Intel Xeon E5-2670)[18]	6419263	1700	3776	364
Acer AW2000h-Aw170h F2 (Intel Xeon E5-2660)[19]	5286503	1275	4146	331
PowerEdge R820 (Intel Xeon E5-4650L)[20]	2790966	457	6102	108

2.1. táblázat

A három típusú szerver számítási képességét (teljesítmény) és energiafogyasztását mutatja be a 2.1. táblázat, mely a SPEC SPECpower_ssj2008 specifikációján alapszik. Meg kell jegyezni, hogy a számítási teljesítmény azaz a ssj_ops [21] (befejezett műveletek száma másodpercenként) és az energiafogyasztás $P_{ac,*}$ 100%-os terheltség mellett mérték, míg $P_{id,*}$ a szerverek energiafogyasztása tétlen állapotban. A 2.1. részben bemutatott két minősítő függvényből származó ($r_e(s)$ és $r_p(s)$) kiszámított sorrendet a 2.2. táblázat mutatja.

Szerver típus	Minősítés teljesítményt alapul véve	Minősítés energiahatékonyságot alapul véve
Intel Xeon E5-2670	$r_p(1) = 1.0$	$r_e(1) \approx 0.64$
Intel Xeon E5-2660	$r_p(2) \approx 0.82$	$r_e(2) \approx 0.66$
Intel Xeon E5-4650L	$r_p(3) \approx 0.43$	$r_e(3) = 1.0$

2.2. táblázat

2.3. Terheltség

Felépítettünk egy I szerverből álló osztályt, így az általunk tekintett modell összesen $I \times J$ szerverből áll ($I=3, J=3$). Ahogy már korábban meg lett említve, a jobok a rendszerhez egy N elemű forrásból érkeznek, valamint a jobok λ paraméterű Poisson-eloszlás szerint generálódnak és kiszolgálási idejük szintén exponenciális eloszlást követ. Minden egyes job 6419253-al (ssj_ops) ekvivalens számítási kapacitást igényel átlagosan, ami azt jelenti, hogy a jobok átlagos kiszolgálási ideje 1 másodperc, ha a jobok az Intel Xeon E5-2670 típusú szerverhez továbbítódnak.

Jelölje μ_1, μ_2, μ_3 az intenzitási rátákat, ha a jobok Intel Xeon E5-2670, Intel Xeon E5-2660 és Intel Xeon E5-4650L típusú szerverhez ütemeződnek. Ebből azt kapjuk, hogy $\mu_1 = \frac{1}{s}$, $\mu_2 = \frac{0.82}{s}$, $\mu_3 = \frac{0.43}{s}$, amit a 2.2. táblázat jól szemléltet.

A 2.1 és 2.2 egyenletekből a teljes rendszer kiszolgálási rátájának értékét kapjuk és egy job kiszolgálásához szükséges átlagos idő: $\mu_{\text{system}} = 6,75$.

3. fejezet

Puffer séma alapú modellek

Ebben a fejezetben a pufferek jellemzőit vizsgálom meg, ami a jobok tárolásáért felelős, amikor nincs elérhető számítási alkotóelem a jobok végrehajtásához. Általában ebben a témában lévő szakirodalmak a számítási rendszereket úgy tekintik, hogy minden egyes szervernek meg van a saját sora a várakozó jobok tárolásához. Ebben a dolgozatban a 2. fejezetben bemutatott általános számítási klaszter szerver, osztály és klaszter szinten felhasznált puffer sémáit vizsgáljuk meg. Ezeket három modellre bontottunk: Seperate Queue, Class Queue és Common Queue. Azt érdemes megemlíteni, hogy a Seperate Queue tekinthető a hagyományos modellnek.

Elsőként a hagyományos Seperate Queue-t mutatom be a 3.1. részben, majd a 3.2. és 3.3. részben a Class és Common Queue-t ismertetem. Ebben a három részben a modellekhez megfelelő implementált ütemezési algoritmusok és a hozzájuk tartozó fontosabb rendszerjellemzők leírása is megtalálható.

A 3.4. részben az energiaszükséglet paraméterekkel kapcsolatos mutatókat írom le, amikor a tétlen szerverek kikapcsolásra kerülnek és amikor nem.

3.1. Separate Queue

A Seperate Queue esetében a modell úgy lett felépítve, hogy minden egyes szerver saját sorral rendelkezik, lásd a 3.1. ábrát. A beérkező jobot az ütemező Algorithm 1 alapján fogja továbbítani a kiszolgáló egységekhez. Az ütemező algoritmus azt a szervert választja ki, ahol a sor a legrövidebb. Ha több tétlen vagy ugyanolyan hosszú sorral rendelkező szerver van, akkor

az ütemező a legmagasabb prioritású (legkisebb indexű) szervert választja. A korábban leírtak alapján 2 féle osztályozást használunk:

- Teljesítmény alapú osztályozás (HP): Ebben az esetben a nagyobb teljesítményű szerverek kapják a legnagyobb prioritást (legkisebb indexet), lásd 2.2. táblázat.
- Energiahatékonyság alapú osztályozás (EE): Ebben az esetben a kisebb fogyasztású szerverek kapják a legnagyobb prioritást (legkisebb indexet), lásd 2.2. táblázat.

A következőekben jelölje c_{ij} az i -edik osztályban lévő j -edik szervert, illetve q_{ij} az i -edik osztályban lévő j -edik szerverhez tartozó puffert. Ekkor c_{ij} a 0 és 1 értékeket veheti fel, ami azt jelöli, hogy az i -edik osztály j -edik szervere foglalt-e vagy sem. A q_{ij} értékkészlete 0-tól $N - I * J$ -ig terjedhet, amely azt mutatja meg, hogy az i -edik osztály j -edik szerverénél található sorban éppen hány igény tartózkodik.

A hálózat állapota t -edik időpillanatban egy $I \times J + I \times J$ dimenziós folytonos idejű Markov-lánccal feleltethető meg:

$$X(t) = (c_{11}(t); \dots; c_{IJ}(t); q_{11}(t); \dots; q_{IJ}(t))$$

Az egyensúlyi eloszlásokat a következőkkel jelöljük:

$$P(c_{11}; \dots; c_{IJ}; q_{11}; \dots; q_{IJ}) = \lim_{t \rightarrow \infty} P((c_{11}(t) = c_{11}; \dots; c_{IJ}(t) = c_{IJ}; \\ q_{11}(t) = q_{11}; \dots; q_{IJ}(t) = q_{IJ}))$$

A fentiek alapján definiált eloszlások miharabbi kiszámításával a rendszer legfontosabb egyensúlyi jellemzőit az alábbi módon lehet kiszámolni:

- R_{ij} – Annak valószínűsége, hogy az i -edik osztály j -edik szervere foglalt

$$R_{ij} = \sum_{c_{11}=0}^1 \dots \sum_{c_{ij}=1}^1 \dots \sum_{c_{IJ}=0}^1 \sum_{q_{11}=0}^{N-I*J} \dots \sum_{q_{IJ}=0}^{N-I*J} P(c_{11}, \dots, c_{ij}, \dots, c_{IJ}; q_{11}, \dots, q_{IJ})$$

- L_{ij} – Annak valószínűsége, hogy az i -edik osztály j -edik szervere tétlen

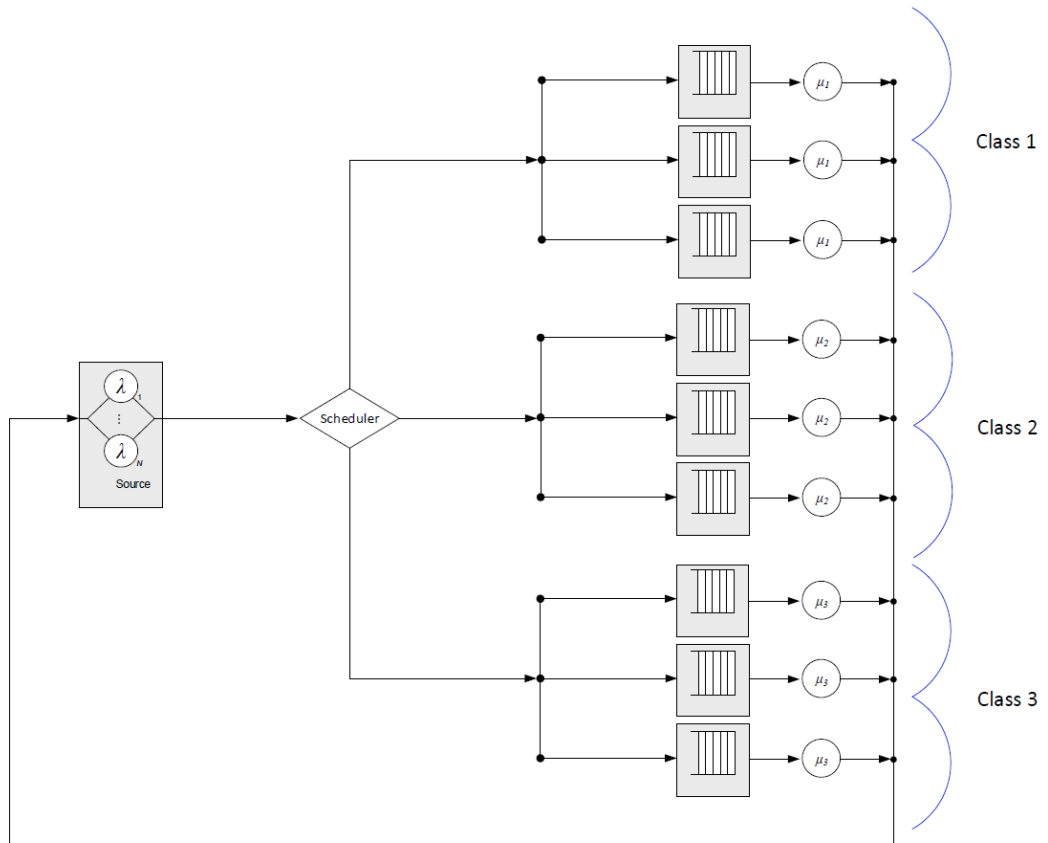
$$L_{ij} = \sum_{c_{11}=0}^1 \dots \sum_{c_{ij}=0}^0 \dots \sum_{c_{IJ}=0}^1 \sum_{q_{11}=0}^{N-I*J} \dots \sum_{q_{IJ}=0}^{N-I*J} P(c_{11}, \dots, c_{ij}, \dots, c_{IJ}; q_{11}, \dots, q_{IJ})$$

- Az átlagos ij -ik sorhossz

$$Q_{ij} = \sum_{c_{11}=0}^1 \dots \sum_{c_{IJ}=0}^1 \sum_{q_{11}=0}^{N-I*J} \dots \sum_{q_{ij}=0}^{N-I*J} \dots \sum_{q_{IJ}=0}^{N-I*J} q_{ij} * P(c_{11}, \dots, c_{IJ}; q_{11}, \dots, q_{ij}, \dots, q_{IJ})$$

- Az átlagos igényszám a sorokban

$$\bar{Q} = \sum_{i=0}^I \sum_{j=0}^J Q_{ij}$$



3.1. ábra. *Seperate Queue modell*

Algorithm 1 A Seperate Queue ütemezési algoritmus

```
1:  $legjobb\_szerver \leftarrow (1, 1)$ 
2: for  $i \leftarrow 1, I$  do
3:   for  $j \leftarrow 1, J_i$  do
4:     if  $szerver(i, j)$  SZABAD then            $\triangleright$  szabad szerver található  $i$ -edik osztályban
5:        $legjobb\_szerver \leftarrow (i, j)$ 
6:       GOTO ÜTEMEZÉS
7:     else
8:       if  $szerver\ sorhossz(i, j) < legjobb\_szerver\ szerverének\ sorhossza$  then
9:          $legjobb\_szerver \leftarrow (i, j)$ 
10:      end if
11:    end if
12:  end for
13: end for
14: ÜTEMEZÉS:  $legjobb\_szerver$  szervernek sorához továbbítsd
```

3.2. Class Queue

A Class Queue esetén minden egyes osztályhoz egy közös puffer tartozik (lásd 3.2. ábra). Jobb egy bizonyos osztályhoz lesznek ütemezve (Algorithm 2 szerint), és ehhez az osztályhoz tartozó pufferben tartózkodnak, amíg ebben az osztályban az összes szerver foglalt. Ha ennél az osztálynál egy jobb távozik bármelyik szerverről, akkor az osztályhoz tartozó pufferből az első várakozó jobb egyből a szabad szerverhez lesz irányítva. Az algoritmus egy érkező jobbot a következő kritériumok szerint fog irányítani: tétlen szerverek, az osztályok prioritása (HP, EE) és a legrövidebb sorhossz.

A következőekben jelölje c_{ij} az i -edik osztályban lévő j -edik szerver, illetve q_i az i -edik osztályhoz tartozó puffert. A c_{ij} 0 és 1 értékeket veheti fel, ami azt jelöli, hogy az i -edik osztály j -edik szervere foglalt-e vagy sem. A q_i értékkészlete 0-tól $N - I * J$ -ig terjedhet, amely azt mutatja meg, hogy az i -edik osztálynál található sorban éppen hány igény tartózkodik.

A hálózat állapota t -edik időpillanatban egy $I \times J + I$ dimenziós folytonos idejű Markov-lánccal feleltethető meg:

$$X(t) = (c_{11}(t); \dots; c_{IJ}(t); q_1(t); \dots; q_I(t))$$

Az egyensúlyi eloszlásokat a következőkkel jelöljük:

$$P(c_{11}; \dots; c_{IJ}; q_1; \dots; q_I) = \lim_{t \rightarrow \infty} P((c_{11}(t) = c_{11}; \dots; c_{IJ}(t) = c_{IJ}; \\ q_1(t) = q_1; \dots; q_I(t) = q_I))$$

A fentiek alapján definiált eloszlások miharabbi kiszámításával a rendszer legfontosabb egyensúlyi jellemzőit az alábbi módon lehet kiszámolni:

- R_{ij} – Annak valószínűsége, hogy az i -edik osztály j -edik szervere foglalt

$$R_{ij} = \sum_{c_{11}=0}^1 \dots \sum_{c_{ij}=1}^1 \dots \sum_{c_{IJ}=0}^1 \sum_{q_1=0}^{N-I*J} \dots \sum_{q_I=0}^{N-I*J} P(c_{11}, \dots, c_{ij}, \dots, c_{IJ}; q_1, \dots, q_I)$$

- L_{ij} – Annak valószínűsége, hogy az i -edik osztály j -edik szervere tétlen

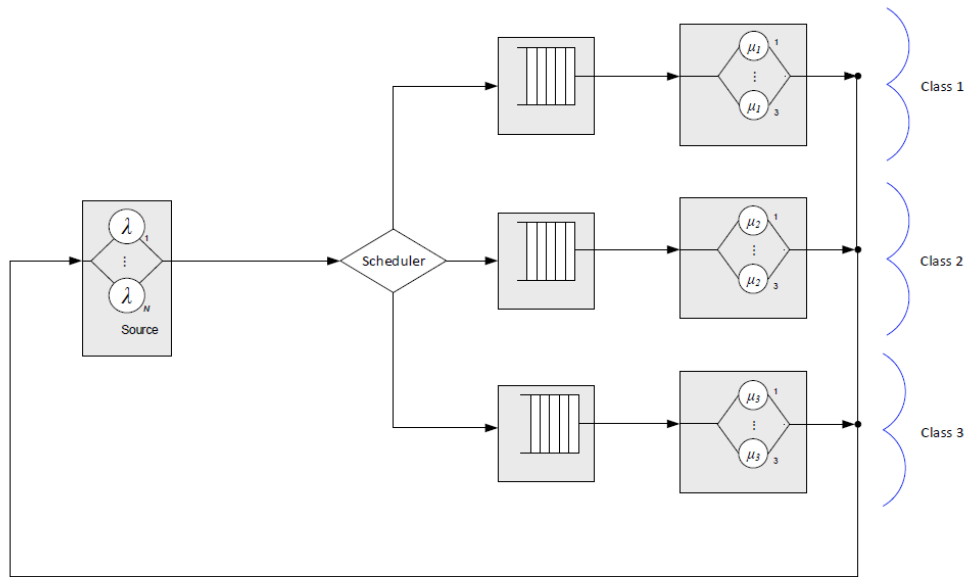
$$L_{ij} = \sum_{c_{11}=0}^1 \dots \sum_{c_{ij}=0}^0 \dots \sum_{c_{IJ}=0}^1 \sum_{q_1=0}^{N-I*J} \dots \sum_{q_I=0}^{N-I*J} P(c_{11}, \dots, c_{ij}, \dots, c_{IJ}; q_1, \dots, q_I)$$

- Az átlagos i -ik sorhossz

$$Q_i = \sum_{c_{11}=0}^1 \dots \sum_{c_{IJ}=0}^1 \sum_{q_1=0}^{N-I*J} \dots \sum_{q_i=0}^{N-I*J} \dots \sum_{q_I=0}^{N-I*J} q_i * P(c_{11}, \dots, c_{IJ}; q_1, \dots, q_i, \dots, q_I)$$

- Az átlagos igényszám a sorokban

$$\bar{Q} = \sum_{i=0}^I Q_i$$



3.2. ábra. Class Queue modell

Algorithm 2 A Class Queue ütemezési algoritmus

```

1:  $legjobb\_osztaly \leftarrow (1, 1)$ 
2: for  $i \leftarrow 1, I$  do
3:   for  $j \leftarrow 1, J_i$  do
4:     if  $szerver(i, j)$  SZABAD then           ▷ szabad szerver található  $i$ -edik osztályban
5:        $legjobb\_osztaly \leftarrow (i, j)$ 
6:       GOTO ÜTEMEZÉS
7:     else
8:       if  $i$ -edik osztály sorhossz  $<$   $legjobb\_osztaly$  osztályának sorhossza then
9:          $legjobb\_osztaly \leftarrow (i, j)$ 
10:      end if
11:    end if
12:  end for
13: end for
14: ÜTEMEZÉS:  $legjobb\_osztaly$  osztályának sorához továbbítsd

```

3.3. Common Queue

Common Queue esetén egyetlen puffer van, ahol a jobok várakozhatnak (3.3. ábra). Ha egy job érkezik, annak a kiszolgálása egyből elkezdődik tétlen szerver(ek) esetén. Ha több mint egy tétlen szerver van az LS a legnagyobb prioritásút (HP, EE) választja Algorithm 3 alapján. Ha LS (Local Scheduler) az összes szerveret foglaltnak találja, akkor a job a közös sorban lesz eltárolva és mindaddig ott vár, míg tétlen nem lesz valamelyik szerver.

A következőekben jelölje c_{ij} az i -edik osztályban lévő j -edik szervert, illetve q_1 a rendszerhez tartozó puffert. A c_{ij} 0 és 1 értékeket veheti fel, ami azt jelöli, hogy az i -edik osztály j -edik szervere foglalt-e vagy sem. A q_1 értékkészlete 0-tól $N - I * J$ -ig terjedhet, amely azt mutatja meg, hogy a rendszer egyetlen soránál éppen hány igény tartózkodik.

A hálózat állapota t -edik időpillanatban egy $I \times J + 1$ dimenziós folytonos idejű Markov-lánccal feleltethető meg:

$$X(t) = (c_{11}(t); \dots; c_{IJ}(t); q_1(t))$$

Az egyensúlyi eloszlásokat a következőkkel jelöljük:

$$P(c_{11}; \dots; c_{IJ}; q_1) = \lim_{t \rightarrow \infty} P((c_{11}(t) = c_{11}; \dots; c_{IJ}(t) = c_{IJ}; q_1(t) = q_1)$$

A fentiek alapján definiált eloszlások miharabbi kiszámításával a rendszer legfontosabb egyensúlyi jellemzőit az alábbi módon lehet kiszámolni:

- R_{ij} – Annak valószínűsége, hogy az i -edik osztály j -edik szervere foglalt

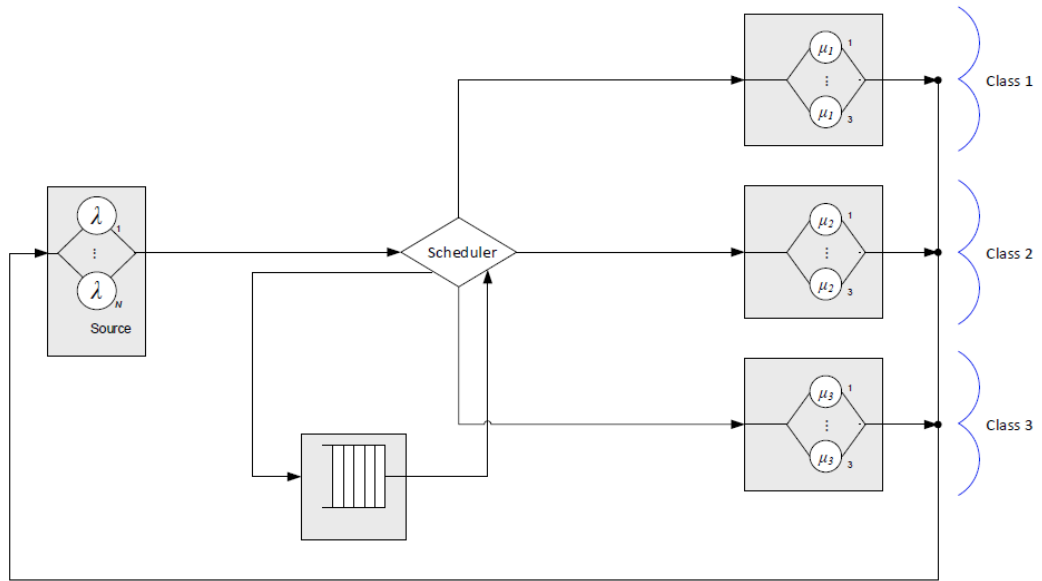
$$R_{ij} = \sum_{c_{11}=0}^1 \dots \sum_{c_{ij}=1}^1 \dots \sum_{c_{IJ}=0}^1 \sum_{q_1=0}^{N-I*J} P(c_{11}, \dots, c_{ij}, \dots, c_{IJ}; q_1)$$

- L_{ij} – Annak valószínűsége, hogy az i -edik osztály j -edik szervere tétlen

$$L_{ij} = \sum_{c_{11}=0}^1 \dots \sum_{c_{ij}=0}^0 \dots \sum_{c_{IJ}=0}^1 \sum_{q_1=0}^{N-I*J} P(c_{11}, \dots, c_{ij}, \dots, c_{IJ}; q_1)$$

- Az átlagos igényszám a sorban

$$\bar{Q} = \sum_{c_{11}=0}^1 \dots \sum_{c_{IJ}=0}^1 \sum_{q_1=0}^{N-I*J} q_1 * P(c_{11}, \dots, c_{IJ}; q_1)$$



3.3. ábra. Common Queue modell

Algorithm 3 A Common Queue ütemezési algoritmus

```
1: for  $i = 1 \rightarrow I$  do
2:   for  $j = 1 \rightarrow J_i$  do
3:     if szerver  $(i, j)$  SZABAD then            $\triangleright$  szabad szerver található  $i$ -edik osztályban
4:        $szabad\_szerver \leftarrow (i, j)$ 
5:       GOTO ÜTEMEZÉS
6:     end if
7:   end for
8: end for
9: ÜTEMEZÉS:
10: if  $szabad\_szerver$  található then
11:   jobot TOVÁBBÍTSD a  $szabad\_szerver$ -hez
12: else
13:   jobot TOVÁBBÍTSD a Common Queue-hoz
14: end if
```

Mindhárom puffer séma esetén a fenti mennyiségek kiszámolásával a következő rendszerjellemzők meghatározása a következőképpen adható meg:

- Az átlagos igényszám a szervereknél

$$\bar{R} = \sum_{i=0}^I \sum_{j=0}^J R_{ij}$$

- A rendszerben lévő jobok átlagos száma

$$\bar{O} = \bar{Q} + \bar{R}$$

- A forrásban lévő átlagos igényszám

$$\bar{N} = N - \bar{Q} - \bar{R}$$

- Átlagos generálási intenzitás

$$\bar{\lambda} = \lambda \bar{N}$$

- *Átlagos tartózkodási idő*

$$\bar{T} = \frac{\bar{O}}{\bar{\lambda}}$$

- *Átlagos várakozási idő*

$$\bar{W} = \frac{\bar{Q}}{\bar{\lambda}}$$

3.4. Energiametrikák

Mivel az (i, j) -edik szerver tétlen és aktív állapotában lévő energiafogyasztását $P_{id,i}$ és $P_{ac,i}$ jelöli, a rendszer átlagos energiafogyasztása, amikor a tétlen szerverek nincsenek kikapcsolva (de tétlen állapotban működve fogyasztanak valamennyi energiát):

$$AE_{no-switch} = \sum_{i=1}^I P_{ac,i} \sum_{j=1}^J R_{i,j} + P_{id,i} \sum_{j=1}^J L_{i,j} \quad (3.1)$$

és a rendszer átlagos energiafogyasztása tétlen szerverek kikapcsolása esetén:

$$AE_{switch-off} = \sum_{i=1}^I P_{ac,i} \sum_{j=1}^J R_{i,j} \quad (3.2)$$

A 3.1. táblázatban a rendszerparamétereket és metrikákat leíró jelölések találhatók összefoglalva.

Jelölés	Paraméter	Érték
I	A szerver osztályok száma	3
J	Osztályokban lévő szerverek száma	3
μ_i	i -edik osztálybeli szerverek kiszolgálási rátája	1; 0,82; 0,43
λ	ilyen paraméterű eloszlás szerint generálódnak a jobok	0,01-0,61
U	A rendszer átlagos kihasználtsága	
$R_{i,j}$	(i, j) -edik szerver foglaltságának valószínűsége	
$L_{i,j}$	(i, j) -edik szerver tétlenségének valószínűsége	
$AE_{\text{no-switch}}$	a rendszer átlagos energiafogyasztása kikapcsolás nélkül	
$AE_{\text{switch-off}}$	a rendszer átlagos energiafogyasztása kikapcsolás mellett	
N	forrásszám	36

3.1. táblázat

4. fejezet

Numerikus eredmények és számolások

A 4.1. részben az elért eredményeket ismertetem a rendszer teljesítményének szempontjából, melyek tartalmazzák az átlagos kiszolgálási, várakozási és tartózkodási időnek alaposabb vizsgálatát.

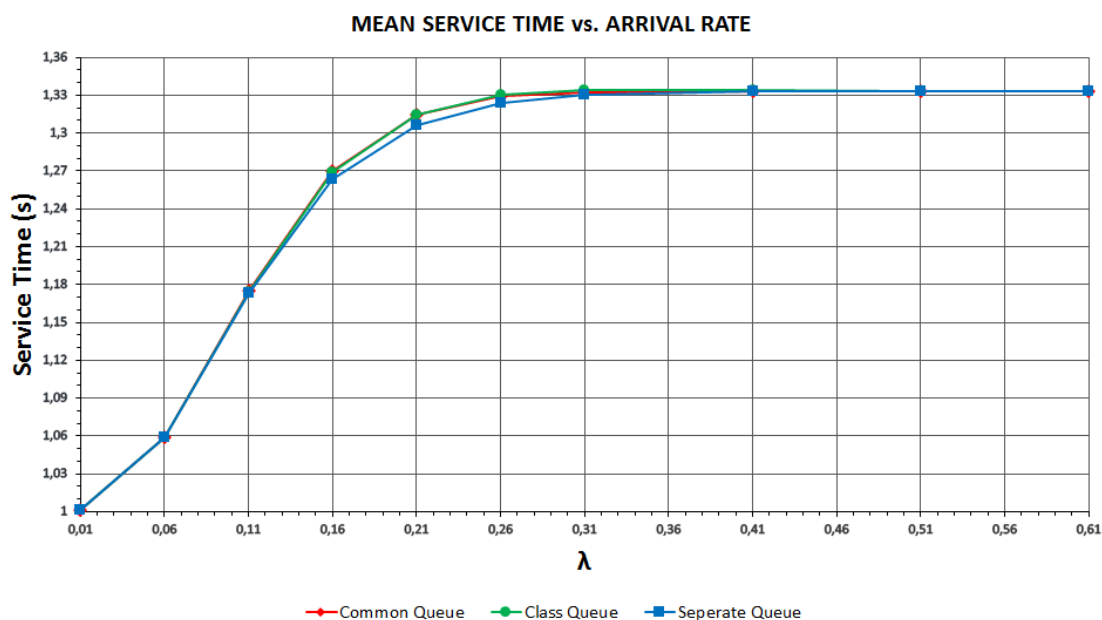
A 4.2. részben pedig azokat az eredményeket taglalom, amelyek 3. fejezetben bemutatott modellek energiafelhasználásának szempontjából fontosak. A rendszer átlagos energiafogyasztása mellett azt a DVFS módszert is megvizsgáltam, amikor a tétlen szerverek kikapcsolásra kerültek a minnél kevesebb fogyasztás elérésének érdekében.

4.1. Rendszerjellemzőkkel kapcsolatos eredmények

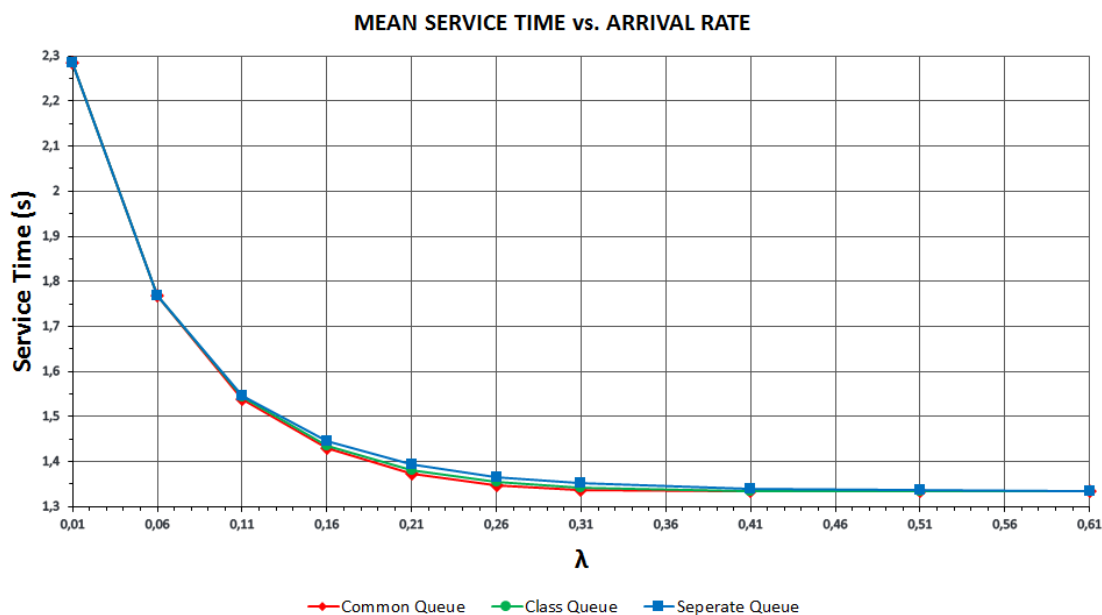
A 2. fejezetben tárgyalt modellek felépítését a MOSEL-2 programcsomagot felhasználva készítettem el valamint a rendszerjellemzők kiszámolása szintén a MOSEL-2 programmal történt. A korábbiakban leírtaknak megfelelően három számítási klasztert definiáltam, valamint mindegyik klaszterben három szerver szerepelt, melyek típus szerint a következők voltak: Intel Xeon E5-2670, Intel Xeon E5-2660 és Intel Xeon E5-4650L (lásd 2.1. táblázat). A MOSEL-2 (Modelling, Specification and Evaluation Language) [22] egy leíró nyelv, mely segítségével különböző programcsomagokat használhatunk, mint például az SPNP-t (Stochastic Petri Net Package) vagy a TimeNet programcsomagot. A MOSEL-2 által szolgáltatott eredményeket grafikusán is ábrázolni tudjuk az IGL (Intermediate Graphical Language) segítségével, mely része a MOSEL-nek.

A teljesítő képesség kiértékeléséhez a várható átlagos kiszolgálási, várakozási és válaszolási idő megfigyelése és kielemezése szükséges. A 4.1. és 4.2. ábra mutatja meg az átlagos kiszolgálási időt az érkezési intenzitás függvényében az összes rendszermodell és elv esetén.

Az figyelhető meg, hogy HP elvnél (4.1. ábra) a beérkezési intenzitás növekedésével az átlagos kiszolgálási idő nő, míg EE elvnél (4.2. ábra) csökken. Ez azzal magyarázható, hogy EE elvnél a jobok először a legkevesebb energiát fogyasztó szerverekhez lesznek ütemezve, amelyek egyben a legkevesebb számolási képességgel rendelkeznek. Így az érkezési intenzitás növekedésével a gyorsabb szerverek is kezdenek terhelődni, tehát a jobok végrehajtásához szükséges átlagos idő is egyre kevesebb lesz ezáltal. HP elvnél pont ennek a folyamatnak az ellentettje igaz, hiszen ott a jobok először pont a leggyorsabb szerverekhez lesznek irányítva. Miután elérkezik az az állapot, amikor azt lehet mondani, hogy majdnem mindegyik szerver teljes mértékben terhelt, akkor az átlagos kiszolgálási idő beáll egy konstans értékre mind HP és EE elv esetén, ahol ez az érték 1,33 körül van. Továbbá az is megfigyelhető, hogy minden egyes sorbanállási séma mellett az átlagos kiszolgálási idő szinte azonos a szerverek terheltségétől függetlenül. Ennek az az oka, hogy ugyanannyi és ugyanolyan szervereket és szerver elhelyezésű struktúrát vonultatunk fel mindhárom modell esetében.

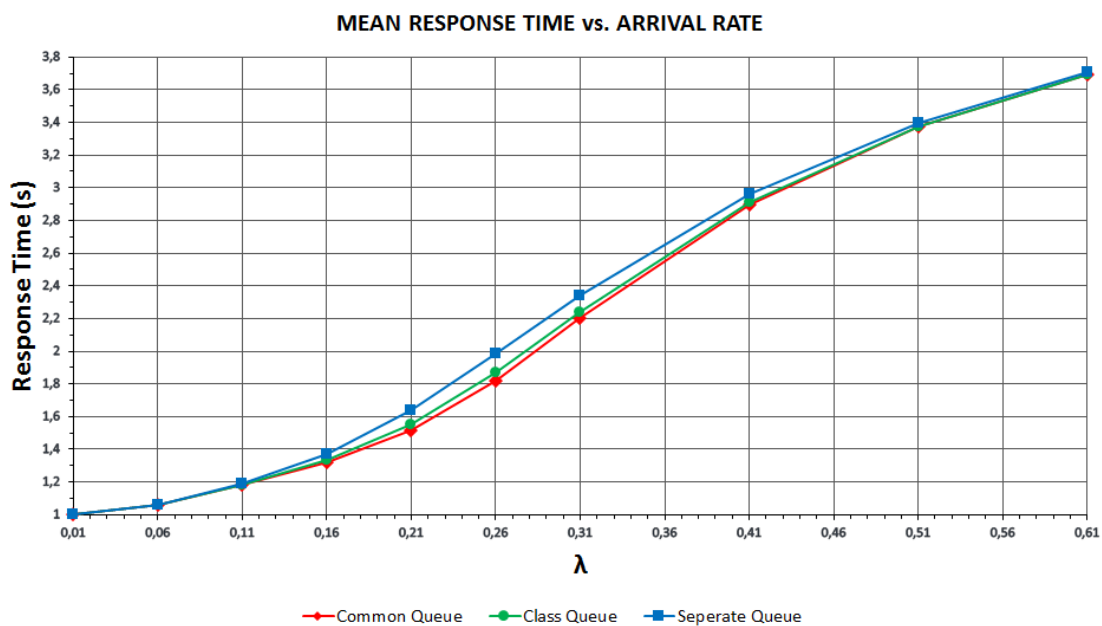


4.1. ábra. Mean service time HP elv alkalmazás mellett

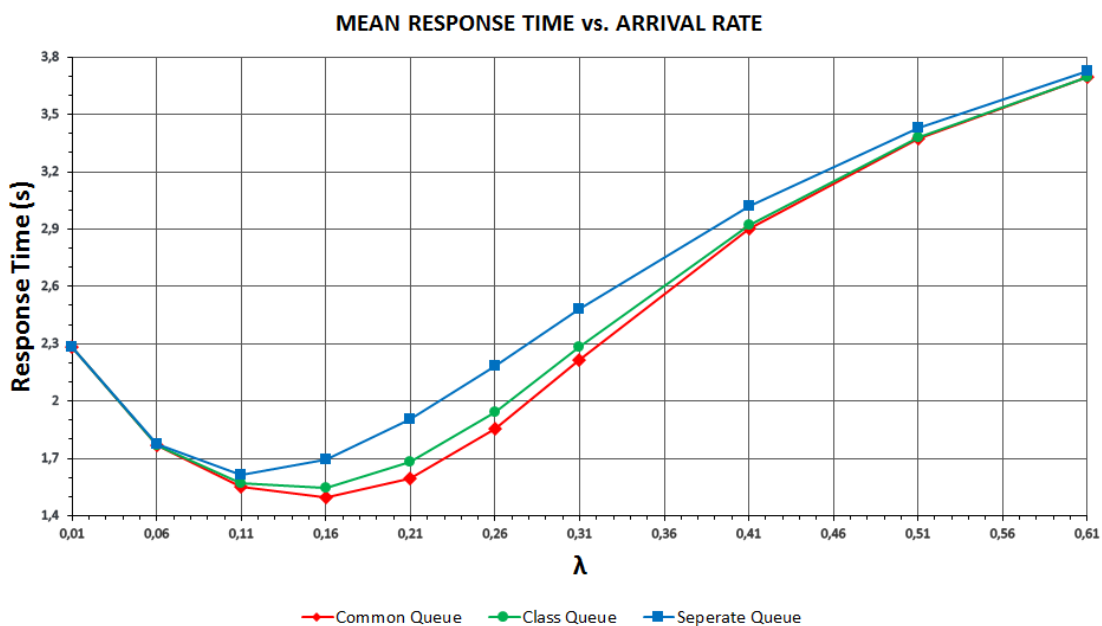


4.2. ábra. Mean service time EE elv alkalmazás mellett

Az átlagos válaszolási időt HP elv esetén a 4.3. ábra, valamint EE elv esetén a 4.4. ábra írja le. Ezeknél az figyelhető meg, hogy a Class és a Common Queue felülmúlja a Seperate Queue modellt a rendszer teljes mértékű leterheltségéig. Az is látható, hogy a Common Queue teljesít a legjobban mind HP mind EE elv esetén. Amíg az intenzitási ráta el nem éri a 0,11-et addig lényeges különbség nem fedezhető fel az alkalmazott modelleknél. Viszont 0,11 és 0,41 között a különbség igen erőteljesen megmutatkozik főleg a Seperate Queue és a másik két modell között. Ezenfelül EE elvnél egy érdekes jelenséget tapasztalhatunk. Ugyanis eleinte az intenzitási ráta növekedése ellenére az átlagos válaszolási idő csökken. Ez azzal magyarázható, hogy EE elvet használva kezdetben a leglassabb szerverek lesznek terhelve, mely magasabb válaszidőket eredményez. A beérkezési intenzitás emelkedésével egyre nagyobb szerepet kapnak a gyorsabb szerverek is és ezáltal a jobok átlagos kiszolgálása is gyorsabbá válik. Ami azt eredményezi, hogy átlagosan kevesebb időt töltenek a rendszerben. Természetesen ez csak egy adott intenzitásig igaz, mert ha már elég nagy a rendszerterheltsége az azt is jelenti, hogy egyre több job van, aminek következtében sorok alakulhatnak ki és nyilvánvalóan a jobok egyre több időt fognak a rendszerben tölteni.



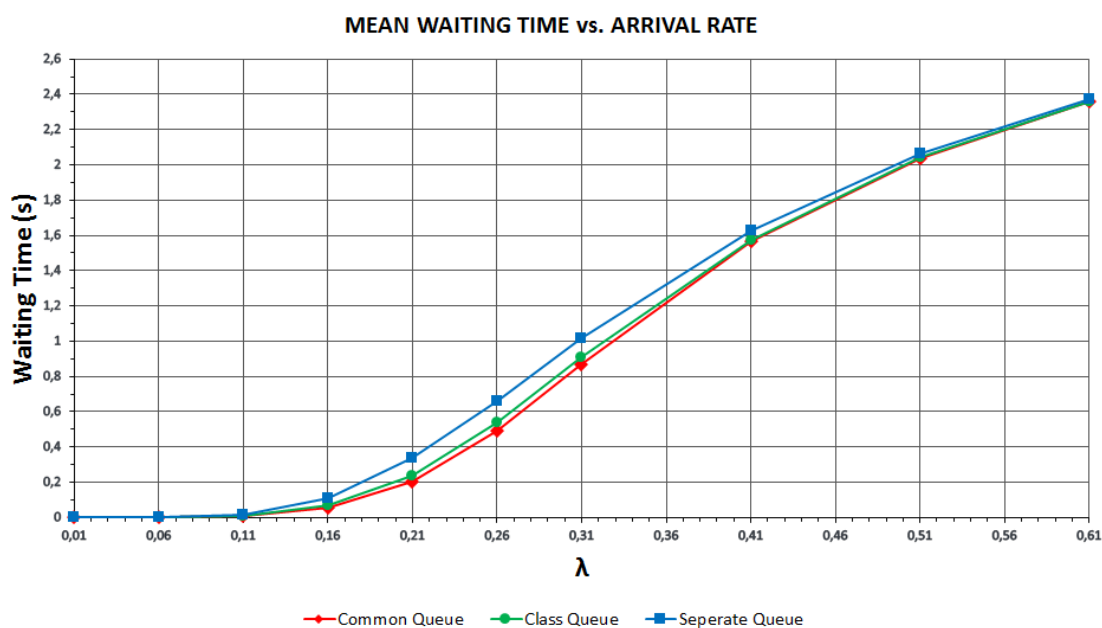
4.3. ábra. Mean response time HP elv alkalmazás mellett



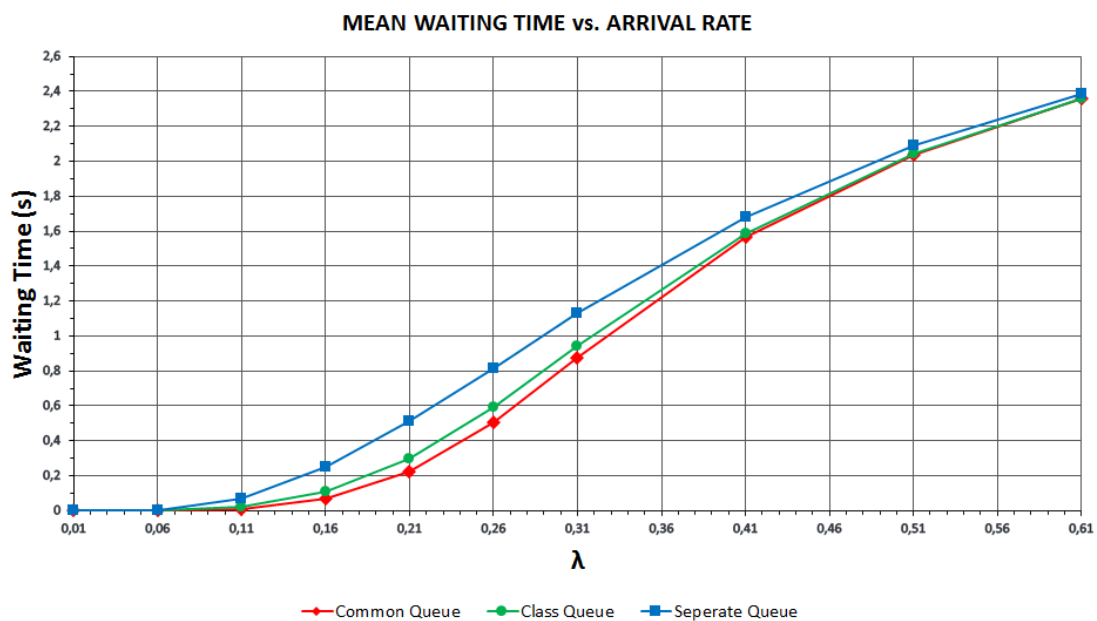
4.4. ábra. Mean response time EE elv alkalmazás mellett

Az 4.5. és 4.6. ábra mutatja az átlagos várakozási időt a beérkezési intenzitás függvényében HP és EE elvet használva. Itt bármelyik elvet nézve azt tapasztalhatjuk, hogy az érkezési intenzitás

növekedése mellett az átlagos várakozási idő is nő. Továbbá mindkét elv esetén a Common Queue teljesít a legjobban és a Seperate Queue esetén az átlagos várakozási idő a legmagasabb a három modell között. Az ok, ami miatt a Common Queue sémánál a legkisebb mind az átlagos válaszolási és várakozási idő az, hogy ez a modell valósítja meg a leghatékonyabb erőforráskihasználtságot, mert ennél nem fordulhat elő az az eset, hogy tétlen szerver esetén várakozó job forduljon elő a sorban. Az is jól megmutatkozik, hogy nagyon kicsi és nagyon nagy terheltség esetén bármelyik elvet is alkalmazva az egyes modellek ugyanúgy viselkednek, hiszen az eredmények is jól mutatják, hogy különbségek ekkor nem tapasztalhatóak.

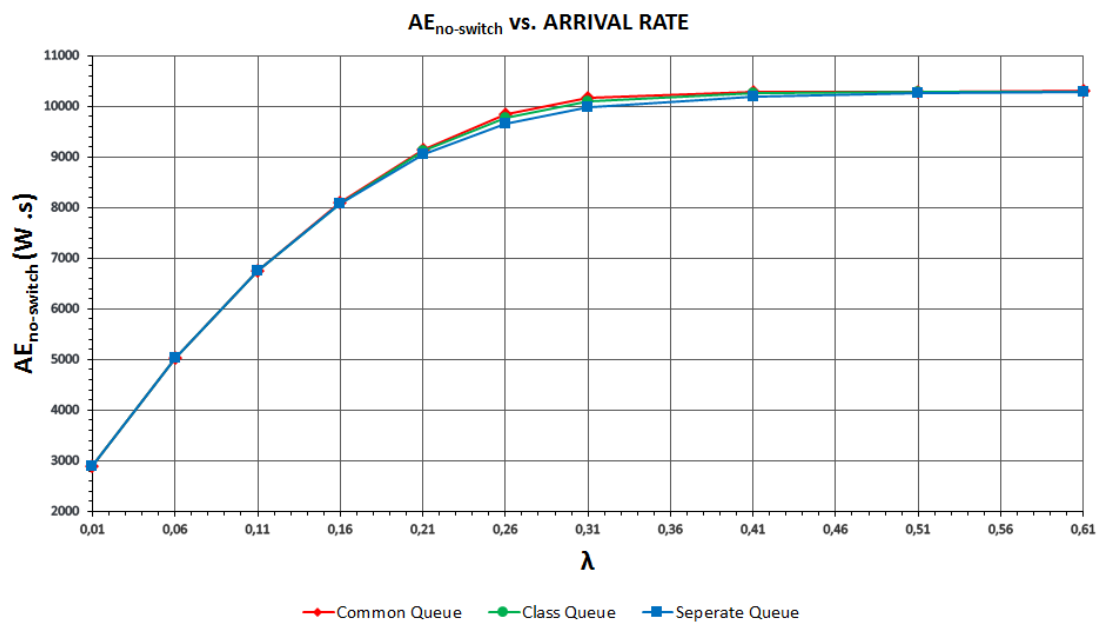


4.5. ábra. Mean waiting time HP elv alkalmazás mellett

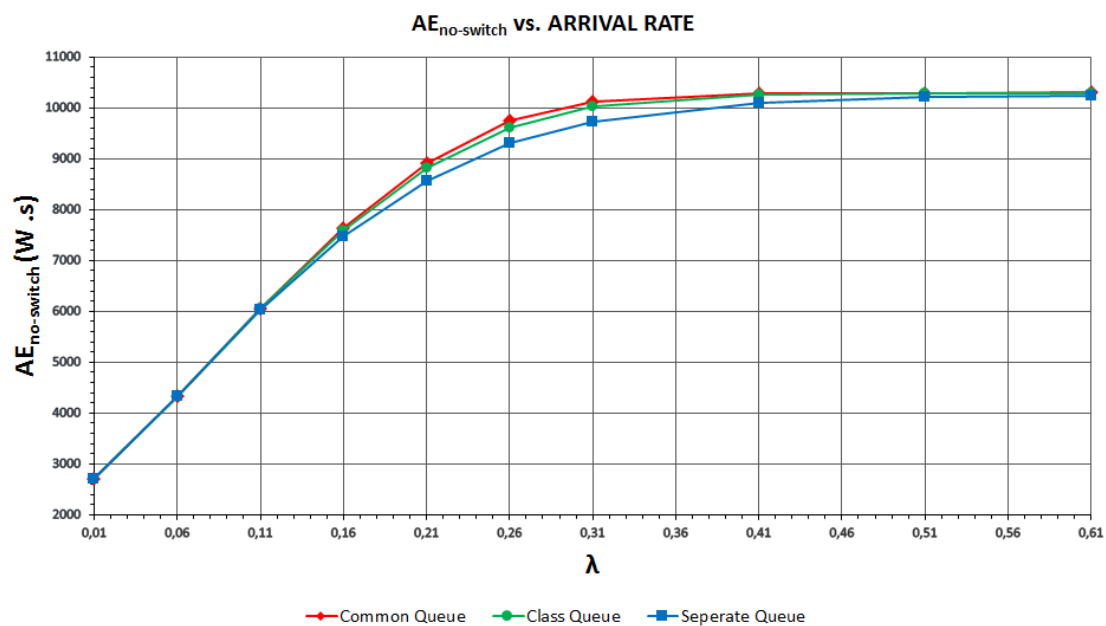


4.6. ábra. Mean waiting time EE elv alkalmazás mellett

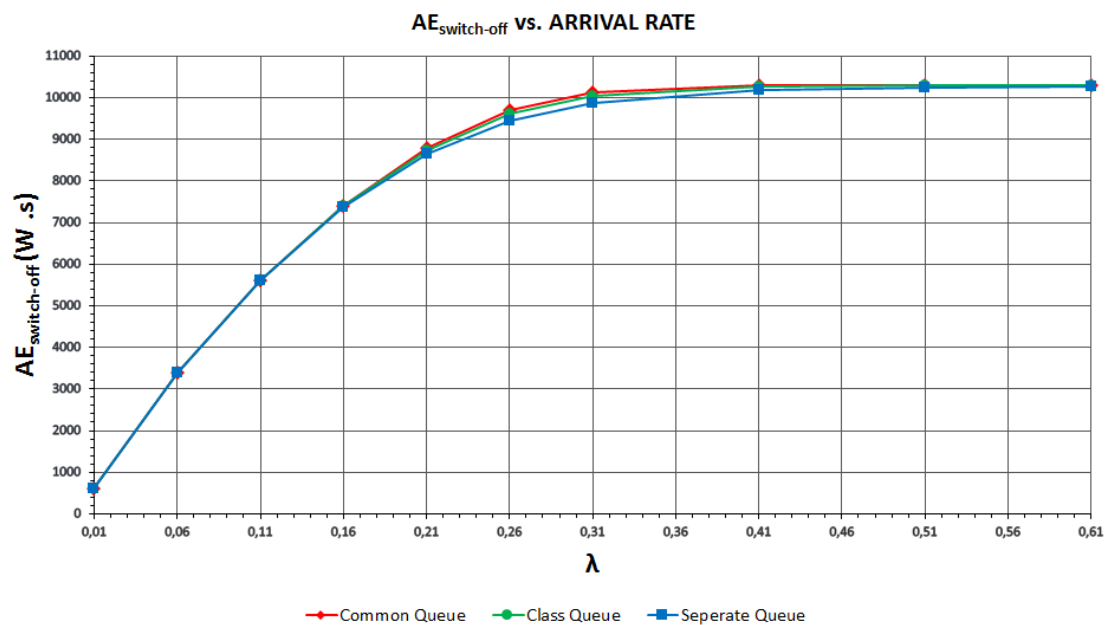
4.2. Energiafogyasztás



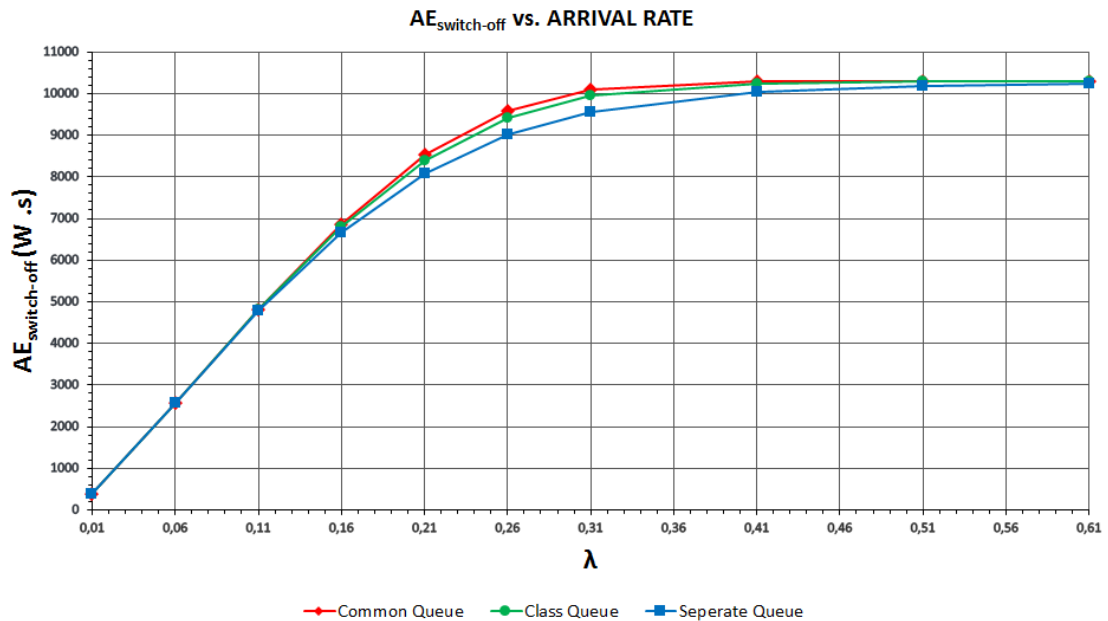
4.7. ábra. Rendszer átlagos fogyasztása HP elv mellett tétlen szerverek kikapcsolása nélkül



4.8. ábra. Rendszer átlagos fogyasztása EE elv mellett tétlen szerverek kikapcsolása nélkül



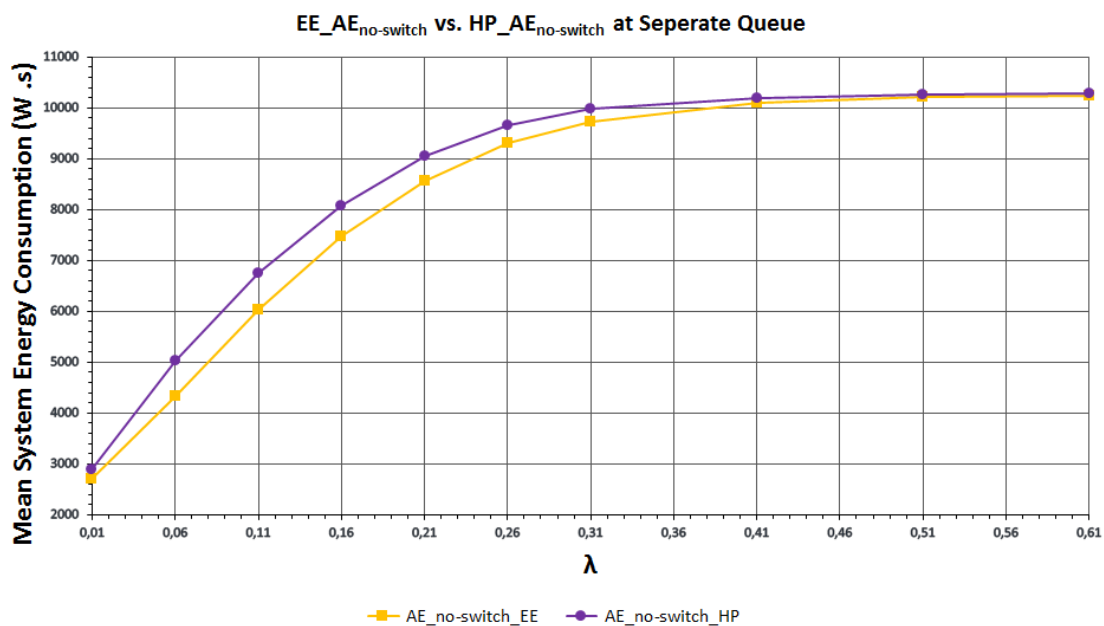
4.9. ábra. Rendszer átlagos fogyasztása HP elv mellett tétlen szerverek kikapcsolása mellett



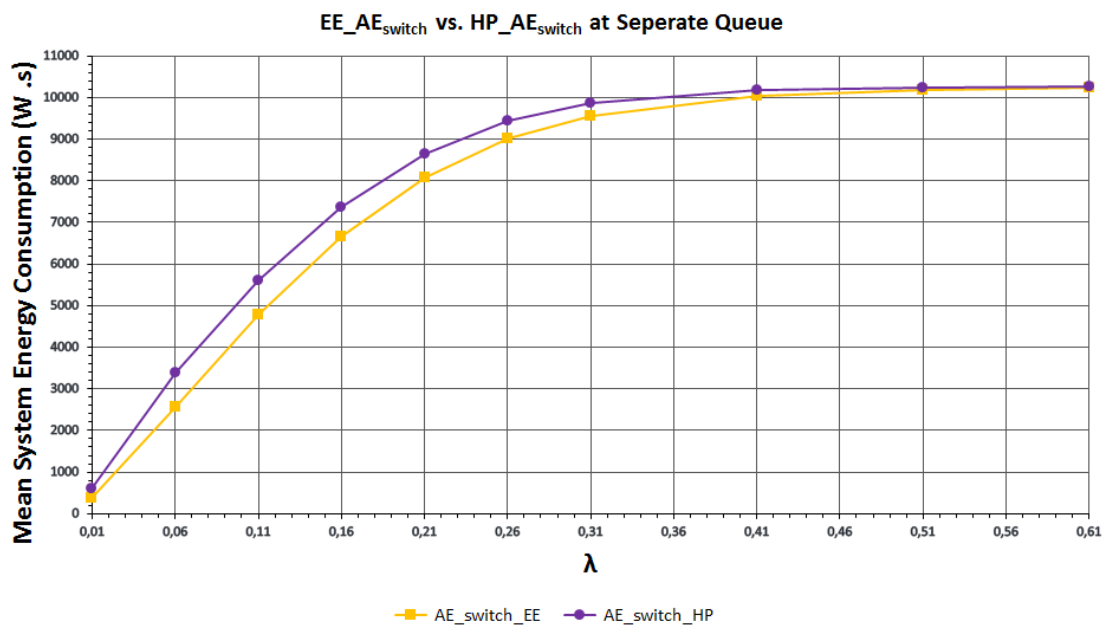
4.10. ábra. Rendszer átlagos fogyasztása EE elv mellett tétlen szerverek kikapcsolásával

A 4.7., 4.8., 4.9. és 4.10. ábrán a rendszer átlagos energiafogyasztása látható abban az esetben, amikor a tétlen szerverek kikapcsolásra kerülnek ($AE_{\text{switch-off}}$), illetve amikor nem kerülnek kikapcsolásra ($AE_{\text{no-switch}}$). Azt érdemes kihangsúlyozni, hogy az energiafogyasztásra a különböző puffer sémák csak viszonylag kis hatással. Amikor a tétlen szervereket nem kapcsolják ki ($AE_{\text{no-switch}}$), akkor a két elv között észrevehető különbséget kis beérkezési intenzitásnál lehet észrevenni, ekkor átlagosan a különbség nagyjából 1000 W.s. Ahogy a beérkezési intenzitás nő úgy csökken a két elv között a különbség és nagy terheltség mellett 10200 W.s körül konvergál mindkettő elv esetén. Ez azzal magyarázható, hogy ha annak valószínűsége, hogy a szerverek foglaltak nagyjából egy, akkor mindkét elvnél az összes szerver dolgozik, így ilyen esetben energiamegtakarítást nem lehet elérni.

Amikor a tétlen szervereket kikapcsolják ($AE_{\text{switch-off}}$), akkor HP és EE elvnél a tendencia azonos, mint amit láthattunk az előző esetben. Itt is amíg a szerverek foglaltsági valószínűsége nem konvergál az 1-hez addig átlagosan a 2 elv között a differencia itt is 1000 W.s körül alakul. Továbbá itt is megfigyelhető, hogy ahogy egyre nagyobb a beérkezési intenzitás úgy csökken a 2 elv közötti energiafogyasztásbeli különbség.

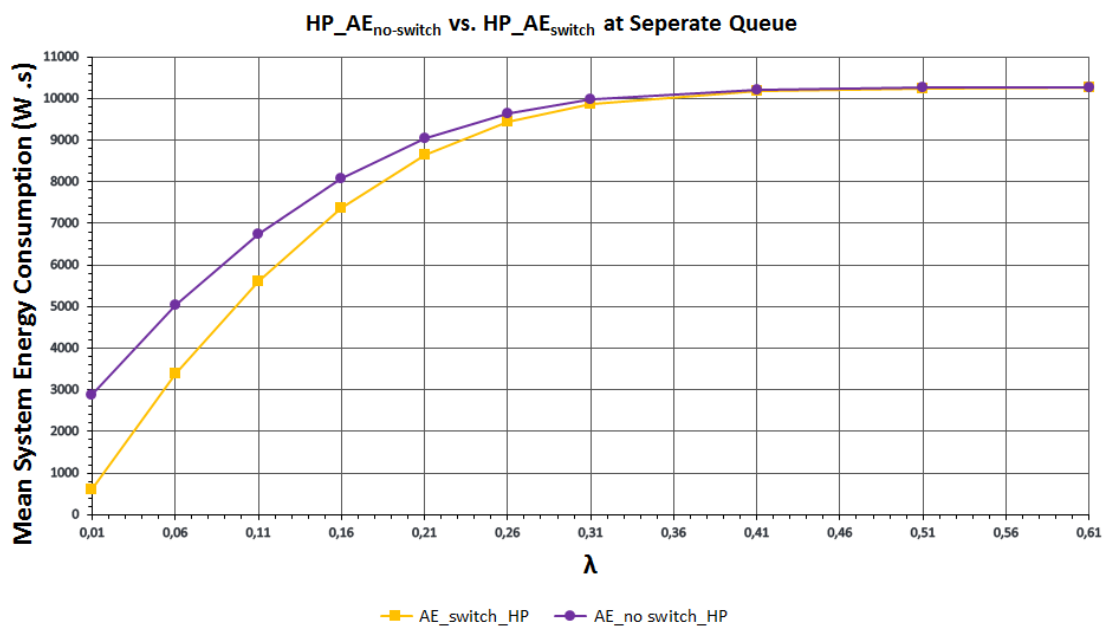


4.11. ábra. Rendszer átlagos fogyasztása HP és EE elv mellett tétlen szerverek ki-kapcsolása nélkül Seperate Queue esetén

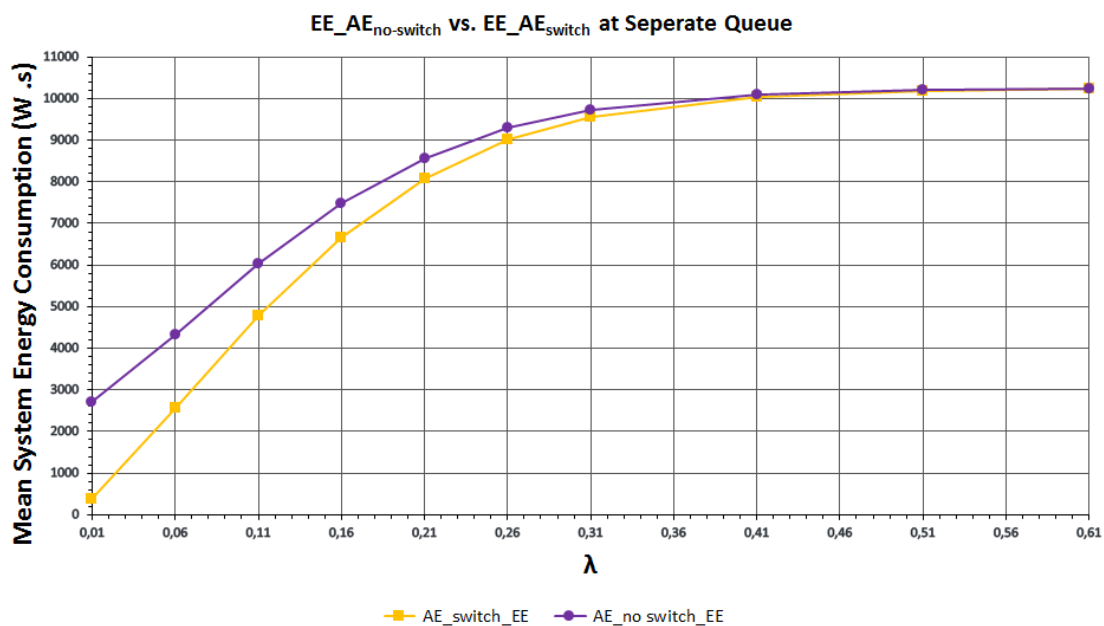


4.12. ábra. Rendszer átlagos fogyasztása HP és EE elv mellett tétlen szerverek ki-kapcsolásával Seperate Queue esetén

A 4.11. és 4.12. ábra mutatja a két elv közötti különbséget tétlen szerverek kikapcsolása ($AE_{\text{switch-off}}$), illetve nem kikapcsolása mellett ($AE_{\text{no-switch}}$) Seperate Queue puffer séma alkalmazásánál. Hasonló ábrákat kapunk Class Queue és Common Queue esetében is. Amint az már korábban is meg lett említve jelentős különbség addig tapasztalható, amíg a rendszer nem telítődik be majdnem teljesen. Nagy rendszerterheltség mellett a 2 elv gyakorlatilag azonosnak tekinthető.



4.13. ábra. Tétlen szerverek kikapcsolásának hatása HP elv esetén



4.14. ábra. *Tétlen szerverek kikapcsolásának hatása EE elv esetén*

A 4.13. és 4.14. ábra szemlélteti a tétlen szerverek kikapcsolásával mennyi energiát lehet megtakarítani. A rendszerterheltség növekedése mellett az energiafogyasztásbeli különbség egyre kisebb lesz. Azt is meg kell jegyezni, hogy a tétlen szerverek kikapcsolása módszer alkalmazása mellett figyelembe kell venni a szerverek visszaállásának idejét az az átmenet időintervallumát, ami egy további időkölséget eredményez. Továbbá nagyon hasonló eredményeket kapunk Common és Class Queue esetén is.

5. fejezet

Összefoglalás

Dolgozatom elején (lásd 1. fejezet) a grid hálózatokról egy rövid áttekintést adtam, amelyeknek a jelentősége az elmúlt években nőtt az egyre több gyorsabb eszköz (pl. processzorok) és a növekvő felhasználószámnak köszönhetően. Ezen felül jellemeztem az ilyen hálózatokat és ismertettem az előnyeiket.

A továbbiakban részleteztem a grid hálózatokhoz való ütemezésének jelentőségét, mivel az elmúlt években egy másik szempont is fontos lett a teljesítmény mellett, ami az energiafogyasztás. Napjainkban a számítógépes grid rendszer hatékony ütemezésén kívül az energiafelhasználása is kiemelt fontosságú lett. Bemutattam az energiafelhasználás csökkentésére használt kétféle módszert: a Static Power Managementet és a Dynamic Power Managementet. Manapság elég népszerűek a DPM-mel kapcsolatos eljárások, melyek közé tartozik a tétlen állapotban tartózkodó szerverek kikapcsolása.

A 2. fejezetben a vizsgált rendszert mutattam be, amely heterogén erőforrású számítási klaszterekből áll, ahol a forrás véges így az általam modellezett rendszer mindig stabil lesz. Legjobb tudomásom szerint véges forrású klaszter hálózatok rendszer jellemzőivel és energiafelhasználásával még nem végeztek kutatásokat, így dolgozatom fontos új eredményeket mutat be. A rendelkezésre álló szervereket három különböző csoportba soroltam és ezen csoportba sorolási módszereket ismertettem, melyek az alábbiak szerint történt:

- Teljesítmény (High Performance)
- Energiafelhasználás (Energy Efficiency)

A 3. fejezetben az alkalmazott puffer sémákat és az ehhez tartozó ütemező algoritmusokat írtam le. Ezenkívül mindhárom puffer séma esetén meghatároztam a folytonos idejű Markov-láncokat illetve a fontosabb rendszerjellemzőket. Ezek után elkezdtem tervezni az így kapott véges forrású klaszter modelleknek a tervezését és kialakítását. Majd a MOSEL-2 programcsomag segítségével elkészítettem a modelleket és szintén a MOSEL-2-t felhasználva meghatározásra kerültek a fontosabb rendszerjellemzők és az energiafogyasztással kapcsolatos értékek.

Ezután a kapott eredményeket el kezdtem kiértékelni és elemezni. A kiértékelés után a következő következtetésekre jutottam: a különböző puffer sémák alkalmazása mellett az átlagos kiszolgálási idő lényegében azonosnak bizonyult mind HP és EE elv esetén. Viszont jelentős eltérések mutatkoztak az átlagos válaszolási és várakozási idők esetén, amikor a beérkezési intenzitás még nem volt nagy. Ugyanis ekkor a hagyományos modelleknél implementált sémánál (Seperate Queue) határozottabban nagyobb értékek jöttek ki és a Common Queue esetében lehetett a legkisebb értékeket tapasztalni. Így az mondható el, hogy a Common és Class Queue határozottabban jobb teljesítményt szolgáltat a Seperate Queue-nál, ezen belül is a Common Queue bizonyult a legjobbnak.

Ezek után megvizsgáltam, hogy milyen hatással vannak a rendszer átlagos energiafogyasztására a három alkalmazott puffer séma. Az eredmények alapján arra a következtetésre jutottam, hogy bármelyik sémát tekintve a rendszer energiafogyasztása nagyjából azonos, tehát nincsenek befolyásolással az energiamegtakarításra. Továbbá kipróbáltam a DPM módszerek közül egyet, amikor is megvizsgáltam, hogy mi történik a rendszer átlagos energiafogyasztásával akkor, ha a tétlen szerverek kikapcsolásra kerülnek. A várakozásoknak megfelelően azt az eredményt kaptam, hogy ezáltal jelentős energiamegtakarítás figyelhető meg abban az esetben, ha a rendszer foglaltságának valószínűsége nem nagy. Viszont ha a beérkezési intenzitás elég nagy, akkor ezzel a módszerrel nem érhető el, hogy a rendszerben lévő szerverek átlagosan kevesebbet fogyasszanak. Emellett a módszerről még az mondható el, hogy a rendszert egy kicsit lassíthatják a szerverek ki/bekapcsolásainak köszönhetően, hiszen ezek a műveletek nyilvánvalóan valamennyi időt igényelnek.

Végkövetkeztetésként az jelenthető ki, hogy csak az energiamegtakarítás szempontjából nézve a vizsgált három séma között szinte nincs különbség, de egy jó puffer séma kiválasztása a teljes klaszter teljesítményét növelni tudja energiafogyasztás többlet nélkül.

Köszönetnyilvánítás

Témavezetőimnek, Dr. Bérczes Tamásnak és Dr. Sztrik Jánosnak, ezúton szeretnék köszönetet nyilvánítani a rengeteg hasznos tanácsért, útmutatásért és az eddig kapott összes szakmai segítségért.

Irodalomjegyzék

- [1] Tran Thi Xuan. Investigate scheduling algorithms. Master's thesis, Budapesti Műszaki és Gazdaságtudományi Egyetem, 2013.
- [2] Tien V. Do, Binh T. Vu, Xuan T. Tran, and Anh P. Nguyen. A generalized model for investigating scheduling schemes in computational clusters. *Simulation Modelling Practice and Theory*, 37(0):30–42, 2013.
- [3] Y. He, W.J. Hsu, and C.E. Leiserson. Provably efficient online non-clairvoyant adaptive scheduling. *IEEE Trans. on Parallel and Distributed Systems*, 19(9):1263–1279, 2008.
- [4] Stylianos Zikos and Helen D. Karatza. The impact of service demand variability on resource allocation strategies in a grid system. *ACM Transactions on Modeling and Computer Simulation*, 20(4):19:1–19:29, 2010.
- [5] Stylianos Zikos and Helen D. Karatza. A clairvoyant site allocation policy based on service demands of jobs in a computational grid. *Simulation Modelling Practice and Theory*, 19(6):1465–1478, 2011.
- [6] Sebastian Szamlewska, Alek Opitz, and Hartmut König. What does grid computing cost? *Journal of Supercomputing*, 6(4):385–397, 2008.
- [7] Stylianos Zikos and Helen D. Karatza. Performance and energy aware cluster-level scheduling of compute-intensive jobs with unknown service times. *Simulation Modelling Practice and Theory*, 19(1):239–250, 2011.
- [8] B. Yagoubi and Y. Slimani. Dynamic load balancing strategy for grid computing. *Transactions on Engineering, Computing and Technology*, 13:260–265, 2006.
- [9] The Standard Performance Evaluation Corporation. Weboldal: <https://www.spec.org/>.

- [10] Ian Foster. What is the grid?, 2002.
- [11] GRID & ATLAS SOFTWARE ILLUSTRATED. Weboldal: <http://www.bellarmine.edu/faculty/amahmood/tier3/SOFTWARE.html>.
- [12] Weizhe Zhang, Hui He, Hongli Zhang, and Mingzeng Hu. Multisite task scheduling on distributed computing grid. *Lecture Notes in Computer Science*, 3033:57–64, 2004.
- [13] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009.
- [14] Ivona Brandic and Schahram Dustdar. Grid vs cloud - a technology comparison. *it - Information Technology*, 53(4):173–179, 2011.
- [15] Vasanth Venkatachalam and Michael Franz. Power reduction techniques for microprocessor systems. *ACM Computing Surveys*, 37(3):195–237, 2005.
- [16] Wissam Chedid and Chansu Yu. Survey on power management techniques for energy efficient computer systems. retrieved from <http://academic.csuohio.edu/yuc/mcrl/survey-power.pdf>, 2002.
- [17] Wissam Chedid, Chansu Yu, and Ben Lee. Power analysis and optimization techniques for energy efficient computer systems. http://web.engr.oregonstate.edu/benl/Publications/Book_Chapters/Advances_in_Computers05.pdf.
- [18] The Standard Performance Evaluation Corporation. Acer incorporated acer aw2000h-aw170h f2 (intel xeon e5-2670). Weboldal: https://www.spec.org/power_ssj2008/results/res2013q1/power_ssj2008-20121212-00590.html.
- [19] The Standard Performance Evaluation Corporation. Acer incorporated acer aw2000h-aw170h f2 (intel xeon e5-2660). Weboldal: https://www.spec.org/power_ssj2008/results/res2012q4/power_ssj2008-20120918-00546.html.
- [20] The Standard Performance Evaluation Corporation. Dell inc. poweredge r820 (intel xeon e5-4650l). Weboldal: https://www.spec.org/power_ssj2008/results/res2012q4/power_ssj2008-20121113-00586.html.

- [21] The Standard Performance Evaluation Corporation. Specpower_ssj2008 result file fields? Weboldal: https://www.spec.org/power/docs/SPECpower_ssj2008-Result_File_Fields.html.
- [22] K. Begain, G. Bolch, and H. Herold. *Practical performance modeling, application of the MOSEL language*. Kluwer Academic Publisher, Boston, 2001.