Tudományos munkafolyamat gráfok alkalmazhatósága dinamikusan változó környezetben

A. Bánáti[1], E. Kail[1], K. Karóczkai[2], P. Kacsuk[2,3], M. Kozlovszky[1,2]

[1] Obuda University, John von Neumann Faculty of Informatics, Biotech Lab
Bécsi str. 96/b., H-1034, Budapest, Hungary
[2] MTA SZTAKI, LPDS, Kende str. 13-17, H-1111, Budapest, Hungary
[3] University of Westminster, 115 New Cavendish Street, London W1W 6UW

{kail.eszter,banati.anna,kozlovszky.miklos}@nik.uni-obuda.hu,
{karoczka, kacsuk}@sztaki.mta.hu

Absztrakt

A tudományos munkafolyamat kezelő rendszerek természetükből fakadóan adatvezérelt jellegűek, melyek számára nem csupán a hatalmas adatmennyiség feldolgozásához szükséges, előre meg nem becsülhető számítási kapacitás jelent nagy kihívásokat, hanem az adatforrások, adattárolók és alkalmazások dinamikus elérése. További problémákat vetnek fel a tudományos munkafolyamat gráfokat futtató, elosztott működést támogató infrastruktúrák (felhő, grid és szuperszámítógépek), melyek esetén különböző számítási-, hálózati- és adattároló-erőforrások válnak elérhetetlenné vagy jelennek meg. Egy ilyen dinamikusan változó környezetben egy munkafolyamat futása során előre nem látható események hatására különböző futási hibákkal találkozhatunk (pl.: hibás futási eredmények, az előírt megszorítások megsértése, futás megszakítása). Ezeknek a hibáknak a nagy része elkerülhető lenne egy olyan munkafolyamat kezelő rendszer segítségével, mely több rugalmasságot biztosít és így az előre nem látható szituációkhoz történő adaptivitást támogatja. A munkafolyamatok értelmezésének és kezelésének dinamizmusa különböző absztrakciós szinteken és a munkafolyamat életciklusának különböző szakaszaiban értelmezhető. Előadásunkat a különböző szinteken és időszakaszokban definiált dinamikus módszerek áttekintésével kezdjük, majd az elosztott számítási infrastruktúrákon futó munkafolyamatok végrehajtása közben bekövetkező hibák vizsgálatával, elemzésével és elkerülhetőségével folytatjuk.

Kulcsszavak: tudományos munkafolyamat, dinamikus munkafolyamat kezelő rendszerek, elosztott rendszerek, elosztott számítás, szuperszámítógép, kivételkezelés, hibák

1. Bevezetés

A tudományos munkafolyamat kezelő rendszerek számítási erőforrásai igen változatos természetűek lehetnek. Akár egy egyedülálló személyi számítógéptől, vagy munkaállomástól, munkaállomások csoportjain át, akár szuperszámítógépes vagy felhő alapú infrastruktúráról is beszélhetünk. Ezekkel a technológiákkal nagy számításigényű és hatalmas adatmennyiséget feldolgozó alkalmazások is futtathatók. Ezen alkalmazások fejlesztése, végrehajtása grid-, szuperszámítógép, vagy felhő alapú infrastruktúrákon igencsak nagy kihívást jelentenek a fejlesztők számára a végrehajtási idő alatt fellépő sokszínű hibahalmazra és a dinamikusan változó környezetre való tekintettel. Előző munkánkban [1] a dinamizmus osztályozásával foglalkoztunk. Egy munkafolyamat életciklusának különböző fázisaiban definiáltuk a dinamizmus szintjeit és a dinamizmussal szemben támasztott igényeket [1. táblázat]. Dinamizmuson egyrészről a rendszernek az előre nem látható, végrehajtási időben bekövetkező váratlan

eseményeire adott válaszadási képességét, másrészről a folyamatosan változó környezethez történő alkalmazkodóképességét, illetve a felhasználói interaktivitás lehetőségét értjük.

A tudományos munkafolyamatok végrehajtása során felmerülő lehetséges problémákat a dinamikus munkafolyamatoknál felépített szintrendszer egyes szintjein vizsgáltuk. Célunk annak felderítése, hogy egy dinamikus rendszer kialakításával és az abban megfogalmazott elvárások alkalmazásával milyen természetű hibák kerülhetők el.

A cikk hátralevő része a következőképpen épül fel. A 2. pontban dolgoztuk fel a témához kapcsolódó szakirodalmat, a 3. szakaszban ismertetjük a mukafolyamatok életciklusának fázisait, valamint az egyes fázisokban egy dinamikus rendszerrel szemben támasztott követelményeket. A 4. részben elemeztük a különböző elosztott számítási környezetekben fellépő hibákat és megoldásaikat a dinamizmus eszközeivel. Cikkünket egy rövid összegzés zárja.

2. A témához kapcsolódó irodalomkutatás

A tudományos munkafolyamatok végrehajtása során előforduló hibák vizsgálata, elemzése, elkerülhetősége széles körben tárgyalt témakör és számos cikk foglalkozik a munkafolyamat-kezelő rendszerek hibatűrő képességével, illetve ennek támogatásával az egyes rendszerekben.

Plankensteiner et al. [2] számítóháló munkafolyamat-kezelő rendszerek (Grid Workflow Management System, GWMS) hibatűrő képességét vizsgálják, illetve áttekintést adnak ennek támogatásáról és megvalósítási mechanizmusairól különböző rendszerek esetén. Munkájukban külön tárgyalják a hiba észlelését, elkerülését, illetve a hibából történő felépülést. Ennek érdekében nagyon mély és részletes rendszerezést adnak a munkafolyamatok végrehajtása közben fellépő lehetséges hibákról, melyet mi is felhasználunk cikkünkben. (lásd később) Hibatűrés növelése érdekében a light-weight és heavy weight checkpoint-ok alkalmazását, az adat és task többszörös tárolását, valamint az alternate task lehetőségét ajálják.

Das és Sarkar [3] számítóhálók (computational grid) esetén ad általános áttekintést a különböző hibatűrő-képességet növelő módszerekről és mechanizmusokról. A hibákat több szinten kezelik, megkülönböztetnek fizikai hibákat, feltétel nélküli befejeződést (például felhasználó által történő leállítás), hálózati hibát, életciklus hibát, processzor hibát és szolgáltatás lejárati hibát. A hibatűrő képességet növelő technikák közül a többszörös tárolást (data and job replication), az ellenőrző pontok (checkpoint) használatát, az ütemezést és a terheléselosztást tárgyalják.

Schroeder és Gibson [4, 5] cikkükben az infrastruktúra (HPC) szintjén készítettek részletes hiba-elemzést és statisztikát. A lehetséges hibák okát típus szerint csoportosítják (emberi hiba, külső tényező keltette hiba (például áramkimaradás), szoftver hiba, hálózati hiba és hardver hiba), ám ezen túl a felmerülő problémákat nem részletezik. Az elemzés eredményeként megállapítják a hibák gyakoriságát, eloszlását, egymáshoz viszonyított százalékos arányát, valamint a hibagyakoriság kapcsolatát az infrastruktúra méretével és életciklusával, illetve kitérnek a hiba utáni felépülés problémájára és időtartamára. Elemzéseik alapjául a Los Alamos National Labs (LANL) szuperszámítógép rendszerének 9 éven át történő monitorozásából származó adathalmaza szolgált.

Mouallem a tézisében [6] szintén a Los Alamos National Labs (LANL) szuperszámítógép rendszerének több éven keresztül gyűjtött adathalmazára támaszkodik, és a lehetséges hibákat, valamint a hibatűrő képességet növelő módszereket tárgyalja, különös tekintettel a Kepler tudományos munkafolyamat kezelő rendszerre. A hibák egymáshoz viszonyított százalékos arányát vizsgálja négy hibacsoport között, hardveres (50%), felhasználói (20%), hálózati-szoftver-környezeti (10%) és ismeretlen (20%), valamint megállapítja, hogy a rendszerekben levő processzorok számának növelésével lineáris arányban nő a meghibásodás is. A futási környezetet három absztrakciós szintre bontja a sikeres futáshoz végrehajtott feladatok szerint: workflow szint mely a workflow vezérlését nyomon követési feladatait látja el, middleware szint, mely a futtatáshoz szükséges szolgáltatásokat biztosítja és a hardware/OS

szint, mely a futtatásért és tárolásért felelős. Megállapítja, hogy a hibák majdnem 70 %-a az utóbbi két szinten történt problémákból fakad. Vizsgálatuk során a munkafolyamatok futtatásakor 9%-ban történt hiba, mely a szuperszámítógépek lefoglalt idejének 7%-át teszik ki.

3. Dinamikus tudományos munkafolyamatok

A tudományos munkafolyamatok életciklusa lebontható elkülönülő fázisokra (hipotézis felállítás, tervezés, példányosítás, futtatás, eredmény elemzés) [7, 8, 9, 10], melyek segítségével a munkafolyamatok létrehozásának, kezelésének és futtatásának a lépései, feladatai és az ezekkel szemben támasztott elvárások áttekinthetőbbé és érthetőbbé válnak.

A ***hipotézis*** felállítása után - melyben a tudós specifikál egy adott tudományos problémát és annak feltételezett megoldását - **a *tervezési fázis*** az első olyan fázis, ami a mi megközelítésünkben részletesebb vizsgálatot igényel. Ebben a fázisban történik a tudományos munkafolyamat megtervezése és összeállítása - amely a hagyományos programozástól sok tekintetben eltér - és az **absztrakt workflow** modell kialakítása (ami vagy egy irányított körmentes gráf, vagy egy leíró nyelvvel definiált program). A modell tartalmazza az egyes feladatokat (task, job) – ezek lehetnek számítási folyamatok, önálló programok, stb - és az ezek közötti adatáramlást. Absztrakt workflow létrehozható újonnan, de előállítható létező (esetleg nyilvános tárhelyről letöltött) munkafolyamatok teljes vagy részenkénti felhasználásával és ezek átalakításával, módosításával. [11, 12] A ***példányosítási fázisban*** az absztrakt workflow-ból létrejön egy **konkrét workflow** (workflow mapping), melynek során megtörténik a munkafolyamat kódjának egy alacsonyabb szintű nyelvre fordítása, a workflow optimalizálása, a paraméterek és az adatok hozzárendelése és az ütemezés is. [11, 12] A ***futási fázisban*** az egyes példányok futtatása következik valamilyen elosztott, nagy teljesítményű infrastruktúrán (szuperszámítógép-rendszer, számítóháló, felhő). Az adott rendszertől függően a végrehajtás részleteiről, paramétereiről és esetleges hibáiról különböző statisztikák és metaadatok (provenance data) gyűjthetők, melyek felhasználásával a többszöri futtatás hatékonysága (ütemezések, különböző döntéshozatali helyzetek és kivételkezelések) növelhetők. Az utolsó fázis, az eredmények ellenőrzése, tesztelése, illetve további provenance adatok és statisztikák gyűjtése és eltárolása.

Előző cikkünkben [1] a dinamikus munkafolyamat-kezeléssel szemben támasztható elvárásokat a három közbülső (tervezési, példányosítási és futtatási) fázisban vizsgáltuk. Az egyes fázisokban (melyek különböző absztrakciós szinteknek is tekinthetők) további szinteket határoztunk meg a problémakör minél mélyebb és részletesebb tárgyalása érdekében. [1. táblázat]

(Megjegyzés: A munkafolyamat végrehajtása alatt a tágabb értelemben vett végrehajtást értjük, azaz a tervezési, példányosítási és futtatási fázist együttvéve. A munkafolyamat futása alatt pedig az egyes task-ok futtatási fázisban történő tényleges futtatását.)

Tervezési fázis: A munkafolyamat „*összeállítási" (composition) szintjén*, a legfőbb támogatás nyelvi vagy fejlett gráf modellezési lehetőségekkel és eszközökkel érhető el. Például egyes munkafolyamat-kezelő rendszerek a feltételes struktúra kialakítását a felhasználóra bízza bizonyos logikai építőelemek összekombinálásával, míg mások a magas szintű nyelvekből ismert IF vagy WHILE struktúrák lehetőségével támogatják a workflow fejlesztőt. [7]

Rendszer szintű támogatást nyújthat a black boxes (fekete-doboz technika), mely lehetővé teszi a task-ok konkrét definiálásának futási időre halasztását. Szintén rendszerszintű dinamizmus érhető el az advance és late modeling technikákkal, mely előbbinél több alternatív végrehajtási „irányt" definiálhatunk, utóbbinál pedig a „fekete-doboz" technikához hasonlóan bizonyos részek és folyamatok definiálását futási időre halasztjuk. [13, 14]

Feladat szintű elvárást, igényt jelenthet a moduláris modelltervezés, illetve az egyes task-ok vagy subworkflow-k újra felhasználhatósága más workflow-k tervezésénél. Ezen lehetőségek gyors és könnyebb modellalkotást tesznek lehetővé.

Példányosítási fázis: Dinamikusan változó környezetben könnyen előfordulhat, hogy például erőforrások kapacitásával és elérhetőségével kapcsolatos, a példányosítás korai szakaszában meghozott statikus döntések a futási fázisára érvényüket vesztik. [15] Ezt kiküszöbölendő dinamikus támogatást biztosíthat az „növekményes fordítás" (incremental compilation) technika [16], mely lehetővé teszi a munkafolyamat részenkénti, időben fokozatos erőforrás kiosztását, illetve futási időben a fordítás dinamikus felülvizsgálatát [17, 18, 19].

A munkafolyamat rugalmas futtatását növelő módszer egy adott workflow-ból több példány létrehozása, melyek száma esetleg nem is ismert a végrehajtás előtt és melyek aztán párhuzamosan is futtathatók. További lehetőséget nyújt rendszer szinten a dinamikus adatkezelés illetve különböző protokollok és biztonsági mechanizmusok támogatása.

Feladat szintű kihívást jelent a „kései adatkötés" (late binding) megvalósítása [20], melynek segítségével a task futási időben vizsgálja meg a bemenő adat helyét vagy helyeit és kiválasztja a legmegfelelőbbet.

Munkafolyamat szintjén a legfőbb dinamikus támogatást az eredeti workflow kisebb részekre, subworkflow-ra bontásával érhetjük el, melyek aztán külön egységként kezelhetők. A részek sorrendje és időzítése a köztük lévő függőségi viszonyoktól függ, akár párhuzamosan is végrehajthatók [10].

A tudományos alkalmazások közül jelentős részt képeznek a hatalmas paramétertéren működő „parameter sweep" alkalmazások, melyek ugyanannak a munkafolyamatnak különböző paraméterekkel történő végrehatását jelentik. [21] Ez a fajta párhuzamosítás gyorsabb végrehajtást és a futási környezet magas szintű rugalmasságát eredményezi.

A végrehajtás teljesítményének növelése a legtöbb esetben hatékony és dinamikus ütemezési algoritmusokkal érhető el, melyek lehetnek task és workflow alapúak is. Provenance adatok felhasználásával az algoritmusok, és ezáltal a rendszer is még adaptívabbá és dinamikusabbá tehető.

Futási fázis: Rendszer szintű támogatás közül a legjelentősebb a kivételkezelés, hiszen ezen a szinten hardveres, hálózati, illetve futási hibák egyaránt bekövetkezhetnek. Ezek kezelését alapvetően két csoportra bonthatjuk. Az egyik, amikor egy nem várt vagy véletlen eseményre a rendszer maga reagál egy támogatott kivételkezelő mechnizmusnak megfelelően. Ez történhet a felhasználó által definiált „alternate task" alkalmazásával, vagy akár provenance adatokon alapuló döntéssel. A másik egy interaktív lehetőség, melynek során a rendszer értesítést küld a workflow fejlesztőnek és felhasználói beavatkozásra vár. Ezen túl egy dinamikus rendszerrel szemben elvárható, hogy ilyen esetekben akár a modellre is kiható, a modellt megváltoztató lépést tegyen.

Szintén rendszer szintű, és általánosan alkalmazott támogatás az ellenőrző, illetve megszakító pontok (checkpoint/breakpoint) elhelyezése a modellben (rendszerfüggő, hogy a tervezési vagy a példányosítási fázisban kell beállítani), mivel hiba esetén a munkafolyamat újraindítása az utolsó ellenőrző ponttól folytatódhat (nem pedig az elejéről), illetve a végrehajtás a megszakító pontoknál megáll és felhasználói beavatkozás után akár más irányban folytatódhat. További kihívást jelenthet, ha breakpoint-ok alkalmazása során a rendszer provenance alapú döntést hozhat a workflow futásának menetéről, vagy esetleg az erőforrások újrakiosztásáról. Itt említendő még a dinamikus erőforrás kiosztás is, mely szintén jelentősen növeli a rendszer rugalmasságát és hatékonyságát.

tervezési fázis **(abstrakt munkafolyamat szintje)**	**rendszer szintű elvárások**	**composition level**	**feladat szintű elvárások**	
	- black boxes - advance and late modelling technique	- language or graph structure	- modularity, reusability	
példányosítási fázis **(konkrét munkafolyamat szintje)**	**rendszer szintű elvárások**	**feladat szintű elvárások**	**munkafolyamat szintű elvárások**	
	- incremental compilation - various protocol support - provenance based sched - multi instance activity	- task based sched. - late binding of data	- partitioning to sub workflows - parameter sweep appl. - wf based sched - mapping adaptation	
végrehajtási fázis **(végrehajtási szint)**	**rendszer szintű elvárások**	**feladat szintű elvárások**	**munkafolyamat szintű elvárások**	**felhasználói szintű elvárások**
	- exception handling - breakpoints - checkpoints - provenance based decisions - monitoring, logging - dynamic resource allocation	- dynamic resource allocation	- alternate task - change the model	- felhasználói beavatkozás

1. Táblázat: A dinamizmus különböző absztrakciós szintjei és a szinteken alkalmazható dinamikus technikák és mechanizmusok

4. Előforduló hibák és elkerülésük dinamikus támogatással

A tudományos munkafolyamatok végrehajtása során felmerülő lehetséges problémákat a dinamizmus korábban definiált különböző absztrakciós szintjein vizsgáltuk. [2. táblázat], [2, 3, 6, 22] A táblázatban az egyes hibalehetőségek után („→” jelölve) feltüntettük egy dinamikus rendszer által megvalósítható lehetséges megoldásukat és kezelésüket. A hibafajták még pontosabb osztályozásához a rendszer szintet további három szintre bontottuk annak megfelelően, hogy az egyes hibák a végrehajtás közben milyen jellegű feladatok ellátása közben merültek fel. Így hardware (HW), operációs rendszer (OS) és middleware (MW) szinteket különböztettünk meg [6].

4.1. Tervezési fázis

Ebben a fázisban rendszer szintű hibákról még nem beszélhetünk, hiszen a munkafolyamat is még csak absztrakt szinten létezik. Task szintű hibákkal sem találkozunk ebben a fázisban, mivel az egyes task-ok bináris kódjait csak a példányosítási fázisban kötjük a konkrét workflow-khoz. Workflow szintjén problémát okozhat, ha a workflow futása során végtelen ciklusba kerül. Erre megoldás lehet egy fejlettebb nyelvi struktúra használata a munkafolyamatok tervezésénél, melyek esetleg magasabb szintű nyelvi elemeket (if, while stb.) is támogatnak. Azonban dinamikus környezetben előfordulhat, hogy a ciklus feltételében megadott változó értéke módosult olyan mértékben, hogy a kiugrási feltétel nem válik igazzá. Ebben az esetben egy felhasználói beavatkozást is lehetővé tevő rendszer nyújtana megoldást.

4.2. Példányosítási fázis

Példányosítási fázisban már rendszerszintű hibák is felléphetnek, melyeket hardware, operációs rendszer és middleware szintjén értelmezhetünk.

A hardware szintű hibák alatt a hardware és a hálózat meghibásodását értjük. Ezek ellen megfelelő redundancia bevezetésével védekezhetünk, ez azonban nem része az általunk vizsgált dinamizmus feltételének, így ezzel a területtel jelen cikkünkben sem foglalkozunk részletesebben. Az operációs rendszer szintű hibákhoz soroltuk a kevés memória vagy lemezterület problémákat, melyek ellen egy provenance alapú ütemezés nyújthat segítséget. Ide sorolhatók még a nem található fájlok, illetve a hálózati torlódás esetén kialakuló hibák is. A middleware szintjén megkülönböztethetünk task beküldési hibákat, hibás hitelesítést, nem elérhető szolgáltatást, illetve fájltárolási hibákat. Task szintjén nem megfelelő formátumú kimeneti adatokról, illetve nem található megosztott könyvtárakról, workflow szinten pedig hibás vagy nem elérhető bemeneti adatokról beszélhetünk. Általánosan is igaz, hogy ezen a szinten a subworkflow alkalmazásának biztosítása és a moduláris, újra hasznosítható workflow tervezés támogatása jelentősen növelheti a rendszer hibatűrését, hiszen e technikák segítségével a workflow kisebb egységekben kezelhető és futtatható. Szintén ehhez a problémakörhöz tartozik a task-onkénti ütemezés lehetősége is.

4.3. Futási fázis

Hardware problémák ebben a fázisban is felléphetnek. Operációs rendszer szintjén hibát okozhat, ha a munkafolyamat futás közben túllépi a megengedett CPU használati időt. A provenance alapú ütemezés ebben az esetben is segíthet. A hibás vagy nem található bemeneti fájlok ellen pedig másolatok készítésével védekezhetünk. Egy dinamikus rendszernek lehetőséget kell nyújtania az adatok több helyen történő tárolására, melyek hálózati torlódás esetén vagy az egyes példányok hiányában egy másik tárolóeszközön lévő másolat felkutatását teszi lehetővé. Dinamikus erőforrás kiosztás megoldást nyújthat olyan esetekben, ha egy task fennakad a helyi erőforrás-kezelő várakozási sorában. Ilyenkor ugyanis mindig csak az aktuális task-ok erőforrás kiosztása történik meg az éppen aktuális információk alapján. Egy task sikertelen lefutásának különböző okai lehetnek, ezen azonban több szinten is megoldást nyújthat egy dinamikus munkafolyamat kezelő rendszer. Egyrészt a sikertelen task helyett egy másik (alternate task), már a tervezésnél megadott task is megoldás lehet, de egy checkpoint beiktatásával akár felhasználói beavatkozással is folytatódhat a munkafolyamat futása. Holtpontok kialakulásának számos feltétele lehet. ha egy ütemezési problémáról van szó, akkor ez esetben a dinamikus erőforrás kiosztás megint csak megoldás lehet. Előfordulhat azonban, hogy tervezési hibáról vagy az adatok egyidejű eléréséről van szó. Előbbi esetben tervezési szinten kell megoldásról gondoskodnunk, illetve felhasználói beavatkozásra kell várni, míg utóbbi esetben a „late binding” technika, azaz az adatok task-okhoz kötésének késleltetése, dinamikussága megfelelő kiutat jelentene. A nem kezelt kivételek esetében pedig egy szélesebb körű kivételkezelés vagy felhasználói beavatkozás szolgáltat megoldást. Felhasználó által előredefiniált, illetve futási időben kikényszerített megszakítás is történhet bizonyos kivételek előfordulása vagy a felhasználó hirtelen beavatkozása esetén.

<table>
<tr><td>tervezési fázis

(absztrakt munkafolyamat szintje)</td><td colspan="3">rendszer szintű hibák</td><td>feladat szintű hibák</td><td colspan="2">kompozició (felhasználói) szintű hibák</td></tr>
<tr><td colspan="3"></td><td></td><td colspan="2">- végtelen kör (ciklus) - dinamikus munkafolyamat esetén
→advanced language and modeling support</td></tr>
<tr><td rowspan="2">példányosítási fázis

(konkrét munkafolyamat szintje)</td><td colspan="3">rendszer szintű hibák</td><td>feladat szintű hibák</td><td colspan="2">munkafolyamat szintű hibák</td></tr>
<tr><td>HW
- gép vagy hw hiba
– hálózati hiba</td><td>OS
- kevés memória →provenance based or dynamic sched.
- kevés lemezterület →provenance based or dynamic sched.
- nem található fájlok
- hálózati torlódás</td><td>MW
- task submission hiba →checkpoint
- hibás hitelesítés →user intervention
- file staging
- nem elérhető szolgáltatás</td><td>- nem megfelelő formátumú kimeneti adat
- hiányzó könyvtárak</td><td colspan="2">- bemeneti adat nem elérhető →data and file replication
- hibás bemeneti adat →data and file replication
- hibás adatátvitel →checkpoint</td></tr>
<tr><td rowspan="2">végrehajtási fázis

(végrehajtási szint)</td><td colspan="3">rendszer szintű hibák</td><td>feladat szintű hibák</td><td>munkafolyamat szintű hibák</td><td>felhasználói szintű hibák</td></tr>
<tr><td>HW
- gép vagy hw hiba
- hálózati hiba</td><td>OS
- CPU idő túllépés →provenance besed, dynamic sched.
- nem található fájlok</td><td>MW
- a helyi erőforráskezelő várakozási sorában fennakadt feladat →dinamikus erőforráskiosztás
- a helyi erőforráskezelő elérése előtt elveszett feladat? →dinamikus erőforráskiosztás + felhasználói beavatkozás</td><td>- job crashed →felhasználói beavatkozás, alternate task, checkpoint
- deadlock/livelock →dinamikus erőforráskiosztás, checkpoint
- uncaught exception (pl. numerikus) →exeption handling + felhasználói beavatkozás</td><td>- hibás adatátvitel →checkpoint</td><td>- User-definable exceptions</td></tr>
</table>

2. Táblázat: Az egyes szinteken előforduló hibák és dinamikus megoldásuk (ha létezik)

5. Összegzés és következtetés

Munkánk során tudományos munkafolyamatok végrehajtása közben, elosztott számítási környezetben bekövetkező hibákkal és elkerülésükkel foglalkoztunk. A tudományos munkafolyamatok életciklusának

ismertetése után a dinamizmus egy korábbi munkánkban meghatározott szintjeinek és elvárásainak áttekintését követve, az összegyűjtött hibákat is a bemutatott szintrendszerbe illesztettük. Megmutattuk, hogy a legtöbbször előforduló hibák egy dinamikus rendszerben milyen eszközökkel kerülhetők el, illetve javíthatók ki. Hardware és hálózati hibákra a dinamikus rendszer nem ad megoldást, de az infrastruktúrába beépített redundanciával az ilyen jellegű hibák száma csökkenthető, továbbá a hiányzó fájlok és szolgáltatások pótlása sem várható el a rendszertől. Összességében azonban elmondható, hogy a felmerülő hibák legalább 70%-ában megoldást nyújtana egy teljesen dinamikus rendszer.

Referencia

[1] E. Kail, A. Bánáti, K. Karóczkai, P. Kacsuk, M. Kozlovszky, Dynamic workflow support in gUSE, MIPRO, 2014 Proceedings of the 37th International Convention

[2] K. Plankensteiner, R. Prodan, T. Fahringer, A. Kertesz, és P. K. Kacsuk, „Fault-tolerant behavior in state-of-the-art grid workflow management systems”, 2007.

[3] A. Das, „On Fault Tolerance of Resources in Computational Grids”, International Journal of Grid Computing & Applications, vol 3, sz 3, o 1–10, szept 2012.

[4] B. Schroeder és G. A. Gibson, „Understanding failures in petascale computers”, Journal of Physics: Conference Series, vol 78, o 012022, júl 2007.

[5] B. Schroeder és G. Gibson, „A Large-scale Study of Failures in High-performance-computing Systems (CMU-PDL-05-112)”, 2005.

[6] P. A. Mouallem és M. Adviser-Vouk, A fault tolerance framework for kepler-based distributed scientific workflows. North Carolina State University, 2011.

[7] E. M. Bahsi, *Dynamic Workflow Management For Large Scale Scientific Applications*, PhD Thesis, B.S., Fatih University, 2006.

[8] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, és J. Myers, *Report on the 2006 NSF Workshop on Challenges of Scientific* „Workflow. Examining the challenges of scientific workflows”, Citeseer, 2006.

[9] E. Deelman és Y. Gil, „Managing Large-Scale Scientific Workflows in Distributed Environments: Experiences and Challenges.”, in e-Science, 2006, o 144.

[10] B. Ludäscher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. D. Roure, J. Freire, C. Goble, és M. Jones, „Scientific process automation and workflow management”, Scientific Data Management: Challenges, Existing Technology, and Deployment, Computational Science Series, o 476–508, 2009.

[11] K. Lee, R. Sakellariou, N. W. Paton, and A. A. A. Fernandes. Workflow adaptation as an autonomic computing problem. In 2nd Workshop on Workflows in Support of Large-Scale Science (Works 07) in Proceedings of HPDC, pages 29–34, 2007

[12] J. Yu és R. Buyya, „A taxonomy of scientific workflow systems for grid computing”, Sigmod Record, vol 34, sz 3, o 44–49, 2005.

[13] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke. A Comprehensive Approach to Flexibility in Workflow Management Systems. In Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration (WACC ’99), pages 79–88. ACM Press, New York, NY, USA, 1999.

[14] M. Pesic: Constraint-Based Workflow Management Systems: Shifting Control to Users, PhD thesis, Eindhoven: Technische Universiteit Eindhoven, 2008

[15] K. Lee, N. W. Paton, R. Sakellariou, E. Deelman, A. A. A. Fernandes, és G. Mehta, „Adaptive workflow processing and execution in Pegasus”, Concurrency and Computation: Practice and Experience, vol 21, sz 16, o 1965–1981, 2009.

[16] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, és J. Good, „Pegasus: A framework for mapping complex scientific workflows onto distributed systems”, Scientific Programming, vol 13, sz 3, o 219–237, 2005.

[17] T. Heinis, C. Pautasso, and G. Alonso. Design and evaluation of an autonomic workflow engine. In 2nd International Conference on Autonomic Computing, pages 27–38. IEEE Computer Society, 2005

[18] R. Duan, R. Prodan, and T. Fahringer. Run-time optimisation of grid workflow applications. In Proc. Intl. Conference on Grid Computing, pages 33–40. IEEE Press, 2006

[19] J.-H. Lee, S.-H. Chin, H.-M. Lee, T. Yoon, K.-S. Chung, and H.-C. Yu. Adaptive workflow scheduling strategy in service-based grids. In GPC, pages 298–309. Springer, 2007.

[20] K. Vahi, M. Rynge, G. Juve, R. Mayani, és E. Deelman, „Rethinking data management for big data scientific workflows”, in Big Data, 2013 IEEE International Conference on, 2013, o 27–35.

[21] P. Kacsuk, K. Karóczkai, G. Hermann, G. Sipos, és J. Kovács, „Supporting dynamic parameter sweep applications in workflows—lessons learnt from the CancerGrid project. PARA’08”, in 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing, Trondheim, Norway, 2008.

[22] R. A. Alsoghayer, Risk assessment models for resource failure in grid computing. Thesis, University of Leeds, 2011.