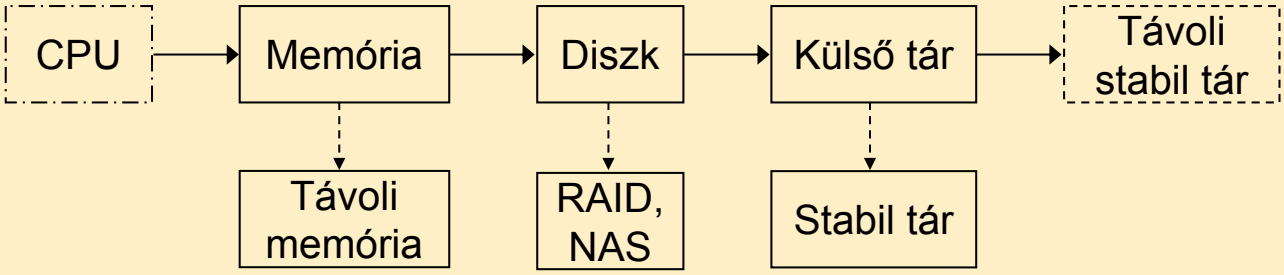
Állapotmentés és helyreállítás

Majzik István
majzik@mit.bme.hu

Motiváció

- Hibaállapot kezelése visszalépő helyreállítással
- Feltételek
  - Komponensek visszaléptethetősége
  - Determinisztikus működés
  - Környezeti interakció külön kezelése
- Jellegzetes használati esetek
  - Hosszú számítási idejű alkalmazások
    - Tudományos számítások
    - Folyamatos működésű alkalmazások
  - „Commodity computing”
    - Skálázhatóság biztosítása olcsó hardverekkel (cloud, grid)
    - Google adat analízis alkalmazások: Átlagban 5 számítógép kiesése egy-egy Map-Reduce job végrehajtása alatt
    - 4000 számítógépből álló hálózat: 6 óránként egy diszk kiesése

Állapotmentési „hierarchia”



Kihívások:

- Komponensek állapotának mentése és visszatöltése
  - Pl. CPU állapot mentése és visszatöltése
- Teljesítmény optimalizálás
  - Megbízhatóság és gyors végrehajtás <-> költségek (pl. cloud)
  - Paraméterek: Replika szám, platform megbízhatóság
- Helyreállítási garancia hosszú idejű számítások esetén
  - Számítási idő >> MTTF esetén csökkenő hatékonyság

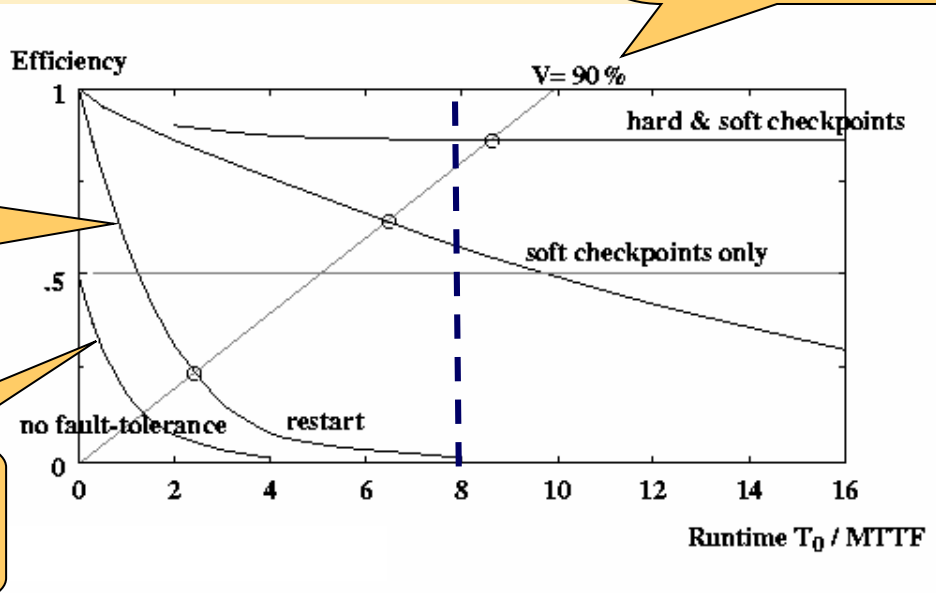
Mentések hatékonysága az egyes szinteken

Itt a hatékonyság (efficiency):

$$\frac{\text{referencia (hibamentes) futási idő}}{\text{hibakezeléssel mért futási idő}}$$

V=90% vonal:
A hibadetektálás 99% hibafedése esetén 90%-os bizalom az eredmények helyességében a vonal feletti területen

99% hibafedésű hibadetektálás van



Újra és újra futtatás, míg ugyanaz az eredmény ki nem jön

Felhasználói állapotmentés: C nyelvű, alacsony szintű megoldás

Checkpoint definíció

- Checkpoint vektor:

```
struct Cpt_vec {
    char *base;
    int len;
} cpt_vec[MAX_VARS];
int cpt_cnt;
```

- Checkpoint állomány:

```
static FILE *cpt_fd;
```

Checkpoint adatok feltöltése

- Elmentendő program adatok (példa):

```
int i;
int results[DIM][DIM];
```

- Checkpoint vektor feltöltése:

```
cpt_cnt = 2;
cpt_vec[0].base = (char *)results;
cpt_vec[0].len  = sizeof(int [DIM][DIM]);
cpt_vec[1].base = (char *)&i;
cpt_vec[1].len  = sizeof(int);
```


Checkpoint adatok kimentése és betöltése

- Checkpoint adatok kimentése

```
fwrite((char *)&cpt_cnt,
        sizeof(int),
        1, cpt_fd);
fwrite((char *)cpt_vec,
        cpt_cnt*sizeof(struct Cpt_vec),
        1, cpt_fd);
for (i=0; i<cpt_cnt; i++) {
   fwrite(cpt_vec[i].base,
          cpt_vec[i].len,
          1, cpt_fd);
}
```

- Checkpoint adatok betöltése:
  - Hasonlóan, **`fread()`** hívással

Processzorállapot mentése és visszaállítása (Unix)

- A **`setjmp()`** és **`longjmp()`** rendszerhívások használhatók: „általános goto”
  - int setjmp(jmp_buf env); <- Állapot rögzítés
  - void longjmp(jmp_buf env, int val); <- Állapotba „ugrás”
- A **`longjmp()`** után a végrehajtás úgy folytatódik, mintha a hozzá tartozó **`setjmp()`** hívásból térne vissza
  - Ez esetben a **`longjmp()`** hívás második paramétere lesz a visszatérési érték
  - Rögzítés és visszaállítás megkülönböztethetők:
    - Sikeres állapot rögzítés: 0 értékkel tér vissza
    - Visszatöltés esetén: a beállított val értékkel tér vissza
- Hasonló funkcionalitás (signal mask mentésével):
  - **`sigsetjmp()`**, **`siglongjmp()`**

Processzorállapot mentése és visszaállítása (Unix)

- Belső állapot rögzítése:

```
jmp_buf st;
setjmp(st);
```

- A rögzített állapot mentése:

```
fwrite((char *)st,
       sizeof(jmp_buf), 1, cpt_fd);
```

- A rögzített állapot betöltése:

```
fread((char *)st,
       sizeof(jmp_buf), 1, cpt_fd);
```

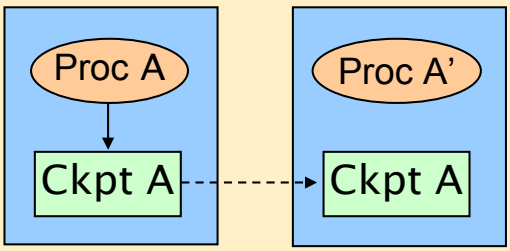
- Visszalépés a mentett állapotba:

```
longjmp(st, 2);
```

Állapotmentés az SAF köztesrétegben (HA middleware), C implementáció

A köztesréteg általnyújtott szolgáltatások

- Checkpoint adatok a fürt szervereinek memóriájában
  - Replikált checkpoint
- Konzisztencia biztosítása
  - Szinkron update
  - Aszinkron update
- Atomi hozzáférés garantálása
- Replikák menedzselése
  - Collocated checkpoint: Elsődleges kijelölése és replika létrehozás
  - Non-collocated checkpoint: Köztesréteg menedzseli a létrehozást
  - Érvényességi idő lejárta után törlés
- Hívások:
  - SaCkptCheckpointOpen, …Close, …Unlink
  - SaCkptCheckpointWrite, …Read
    - IoVector: Specifikálja a mentendő/helyreállítandó memóriatartományt
    - NumberOfElements: Bejegyzések száma az IoVector-ban
    - Bejegyzések részei: sectionId, dataBuffer, dataSize, dataOffset

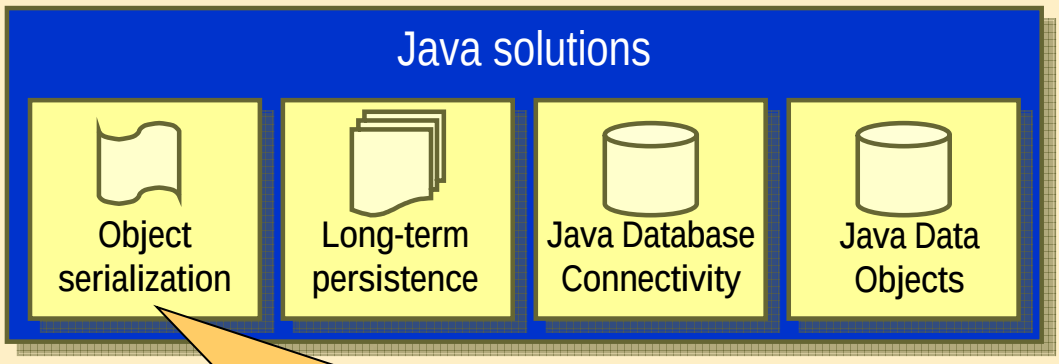


Állapotok (objektumok) mentése J2SE alkalmazásokban

Java alapú megoldások

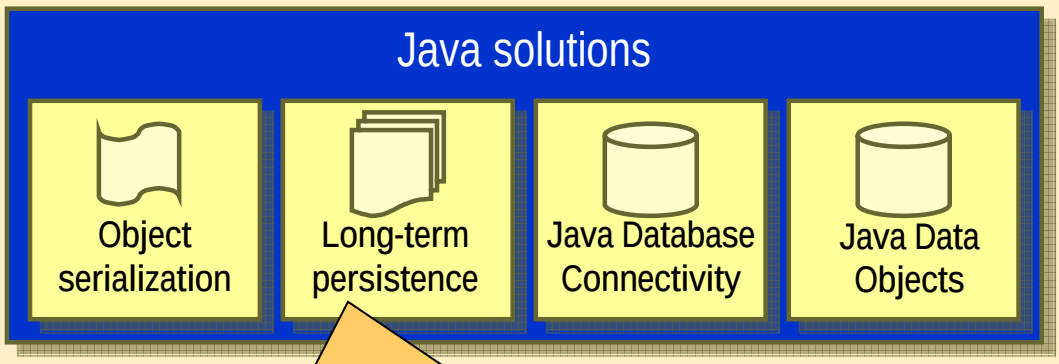
- Object serialization
  - Objektum gráfok tárolása binárisan (stream-ekben)
- Long-term persistence
  - Objektum gráfok tárolása XML fájlokban
- Java Database Connectivity (JDBC)
  - SQL beágyazása Java forrásokba
- Java Data Objects (JDO)
  - Automatikus felműszerezés DB rutinokkal

Megoldások jellegzetességei és előnyei



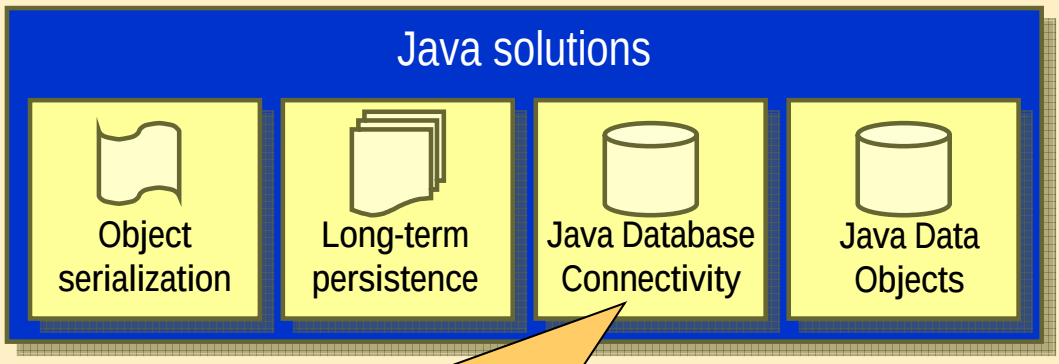
Bináris bájtsorozattá alakítás és mentés
- Serializable interfész implementálása
- ObjectOutputStream writeObject metódusa kiment ObjectInputStream readObject metódusa beolvas
- Automatikus, de bináris (hordozhatóság gond lehet)
- Testreszabás: Externalizable interfész
  - writeExternal, readExternal metódusok

Megoldások jellegzetességei és előnyei



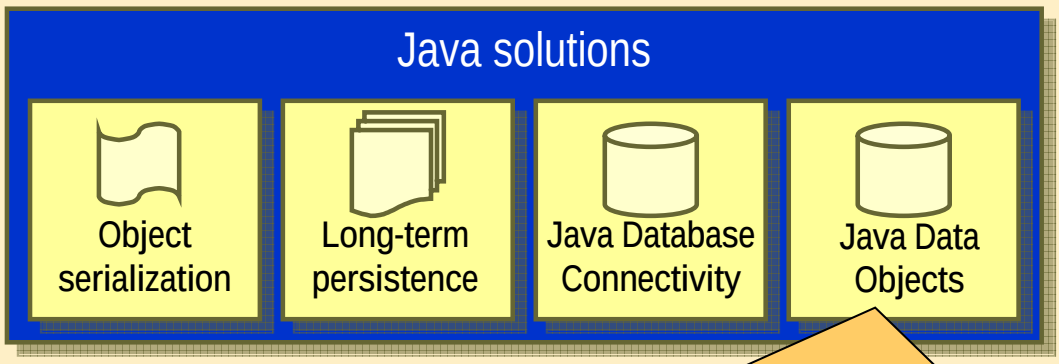
XML állományba történő mentés
- JavaBean konvenciók betartása (getter/setter)
- XMLEncoder osztály writeObject metódusa kiment XMLDecoder osztály readObject metódusa beolvas
- Automatikus, de nagy méretű XML (nem mindig)
- Testreszabás: PersistenceDelegate osztályok

Megoldások jellegzetességei és előnyei



Adatbázisba mentés beágyazott SQL alapon
- Objektumok explicit kiírása adatbázis táblába
- Teljes kontroll a definiálás, kiírás, olvasás felett
- Kiírás és beolvasás SQL utasításokkal
  - createStatement, executeUpdate, executeQuery
- Kézi, de teljesen optimalizálható megoldás
- Testreszabás: SQL szinten

Megoldások jellegzetességei és előnyei



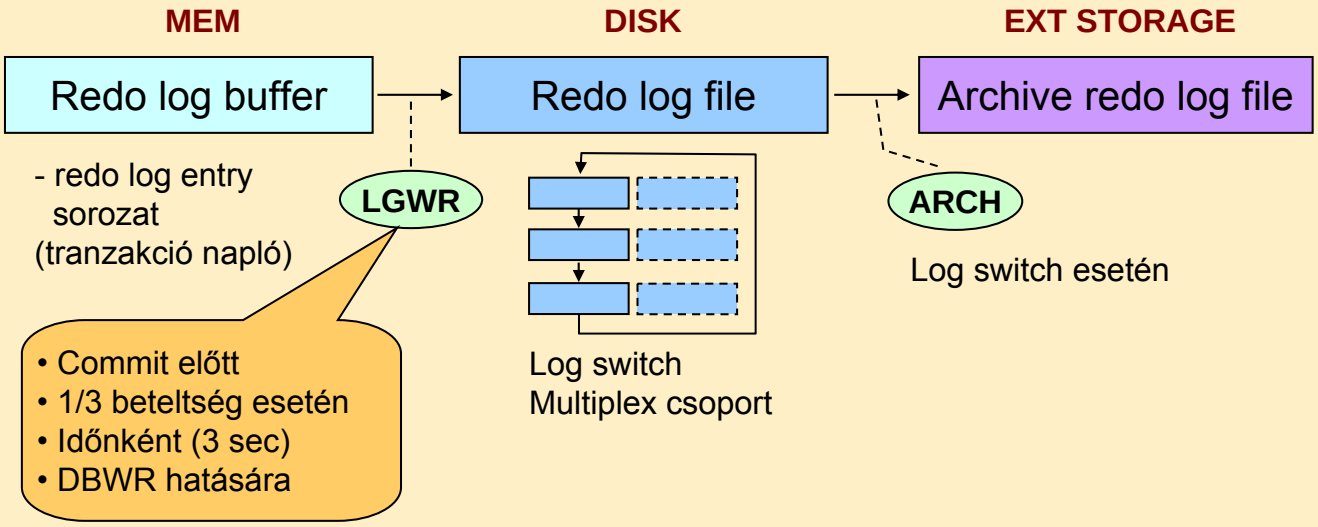
Konfiguráció és (automatikus) kiegészítés mentéshez
- Perzisztencia tulajdonságok megadása XML leíróban (pl. mező hossz adott attribútumhoz)
- PersistenceCapable interfész megvalósítása szükséges
- Tranzakció jellegű mentés és beolvasás
- Kiegészítés (felműszerezés) a tényleges mentéshez és visszaolvasáshoz
  - Forráskód vagy bájtkód szinten
- Automatikus megoldás lehet (implementációfüggő)

Állapotmentés és helyreállítás adatbázis-kezelő rendszerekben

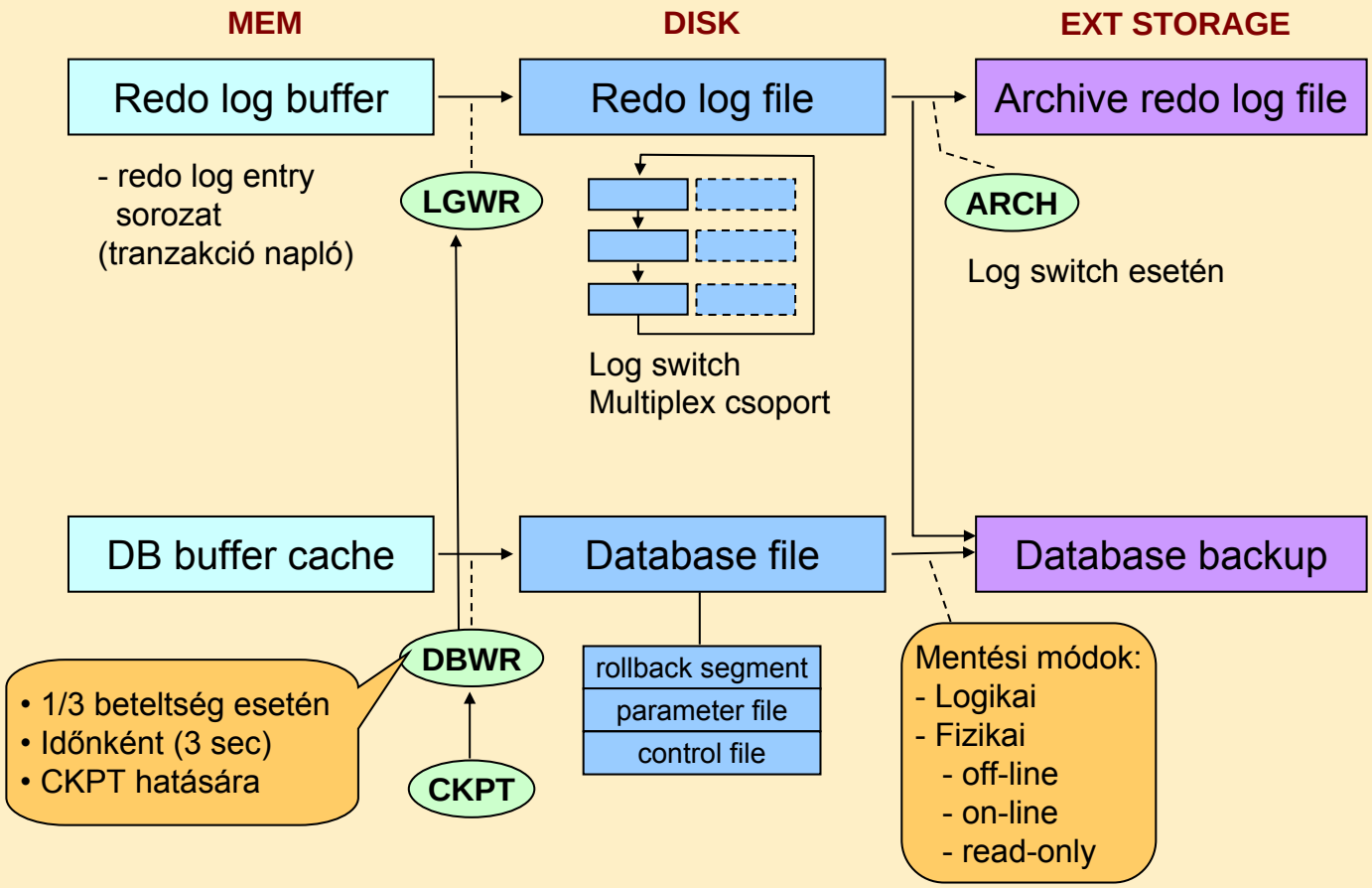
Mentések

- Mentendő adatok:
  - Tábla adatok: Adatbázis cache, adatbázis táblák
  - Rollback szegmensek
    - Abortált tranzakciók általi módosítások visszavonásához
  - Redo log
    - Műveletek naplózása újrajátszáshoz
  - Control információk
    - System area
    - User area
- Mentési hierarchia:
  - Mem → Disk → External (stable) storage

Mentési hierarchia I.



Mentési hierarchia II.



Helyreállítás

- Hibadetektálás:
  - SMON: rendszerfolyamatok felügyelete
    - SGA: System Global Area (locking, status, pool, …)
  - PMON: felhasználói folyamatok felügyelete
    - PGA: Program Global Area (stack, session memory)
  - Operátor
- Lépések a helyreállításhoz:
  - Adatbázis táblák visszatöltése a Database backup alapján
  - Roll forward (redo log alapján)
  - Rollback (rollback szegmensek alapján)
- Mitől függ a mentési stratégia?
  - Adat kritikusság (fizikai / logikai mentés)
  - Adatbázis használat (on-line / off-line mentés)
  - Adatbázis hozzáférés gyakoriság (fizikai off-line / logikai mentés)