Performance and Fairness for Users in Parallel Job Scheduling

Dalibor Klusáček[1,2]
Hana Rudová[2]

[1]CESNET, Czech Republic
[2]Faculty of Informatics, Masaryk University, Czech Republic

16th Workshop on Job Scheduling Strategies for Parallel Processing,
IPDPS, Shanghai, China, 25 May 2012

Motivation

- Efficient application of metaheuristics in Grid/cluster job scheduling
- Scheduling in Czech NGI MetaCentrum
  - Managed by queue-based scheduler
    - PBS-Pro, TORQUE (a form of Backfilling)
- Problems of interest
  - **Performance**
    - Wait time, slowdown, response time
  - **Fairness**
    - To keep users satisfied
  - **Scheduler's behavior** – users keep asking:
    - “Why my job has not started yet?”
    - “Why my job waits when there are free resources?”

Current Approaches

- PBS, LSF, SGE, TORQUE, …
  - Mostly (aggressive) backfilling
    - No reservations vs. EASY backfilling vs. Conservative backfilling
    - Decisions made in an ad hoc fashion
  - **Fairness** is very important
    - FCFS somehow fair but inefficient
    - EASY backfilling is dangerous – large jobs may be delayed
    - Conservative backfilling – quite fair as no job can be delayed
    - Prioritized queues by fairshare principles (balance the user's share)
  - Predictability is not usually supported
    - Advance Reservations may degrade performance
    - Cons. Backfilling is not widely used (reservations limit backfilling opportunities)

Contribution

- Realistic application of metaheuristics in Grid/cluster job scheduling
  - Flexible behavior – based on applied criteria and current situation
- Real-life based problem and goals
  - Large problem instances
  - **Performance**
  - **Fairness**
  - **Fast solution** (limited runtime awareness)
  - Toward predictions
- Further work
  - Prototype implementation in actual scheduler (TORQUE)

What is “fairness”?

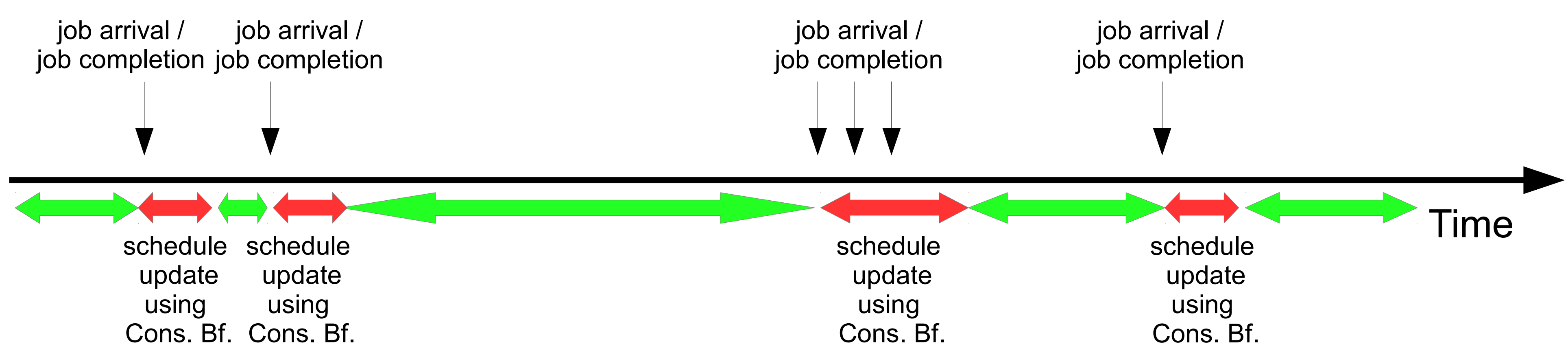
- Inspired by the **fairshare setup** used in MetaCentrum
  - Maximize the share of mostly “penalized” user
  - Prioritizes users with **lower resource consumption**
  - Prioritizes users with **higher wait time**
- Basic principles
  - Fairshare priority = normalized user wait time (NUWT)
    - NUWT = $\frac{user\ wait\ time}{user\ CPU\ time}$
    - NUWT = “how many seconds user waits for one second of job execution”
  - Balancing NUWT values
    - Decreases the differences in the performance delivered to the users

Proposed Approach

- Combination of known “best practices”
  - Use **Conservative backfilling**
    - Conservative backfilling → every job gets a reservation
    - Reservations → fairness (no “unlimited” delays)
    - Backfill-like approach (efficient utilization)
    - Predictability – plan of job execution
  - Use **optimization**
    - Improve quality of execution plan (job schedule)
    - Subject to schedule evaluation → identification of inefficiencies
      - Wait time
      - Bounded slowdown
      - Response time
      - Fairness

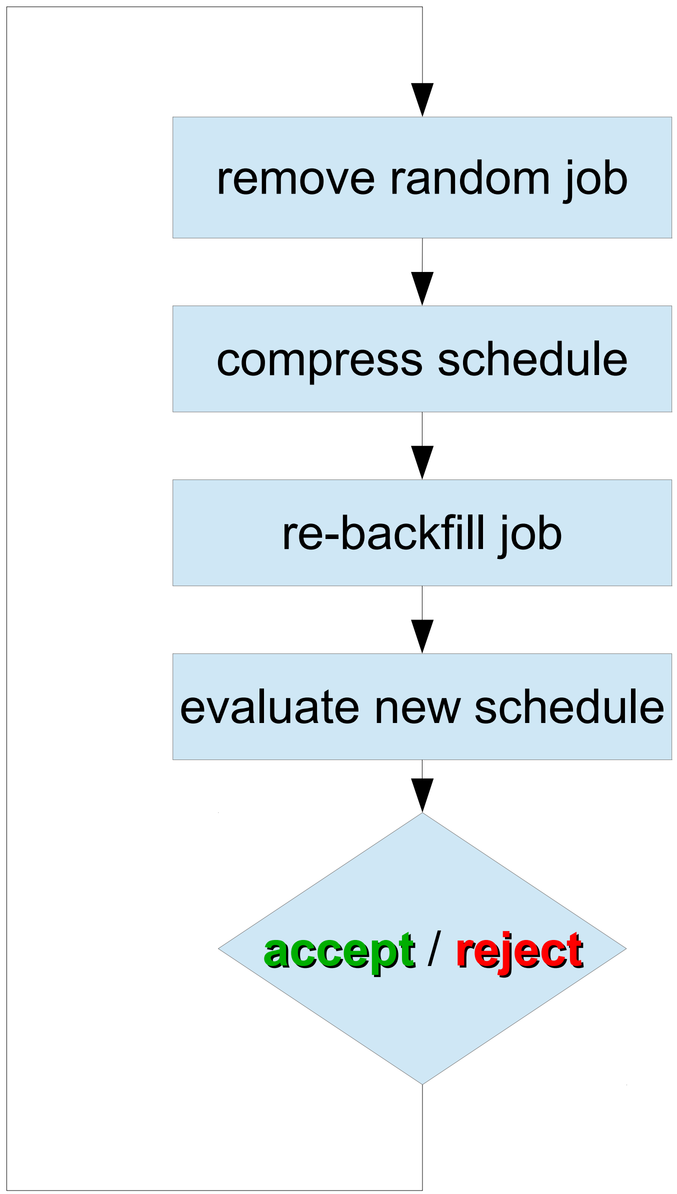
Optimization – limited runtime

- Metaheuristics can be time consuming
  - Limited time due to the on-line problem character
- Time-efficient approach
  - (Valid) initial schedule created quickly using Conservative Backfilling (see)
  - Optimization is only executed when there are no higher priority events such as job arrivals or job completions (see depicting available time)
  - Optimization can be stopped after each iteration when necessary



Optimization – Tabu Search

- Improves initial scheduled delivered by Conservative backfilling
- Tabu search-inspired optimization algorithm (TS)
  - Tabu list prevents short cycles
  - Selective re-backfilling guided by evaluation
- Evaluation
  - Guides the optimization phase
  - Performance and Fairness related criteria
    - Wait time
    - Bounded slowdown
    - Response time
    - NUWT



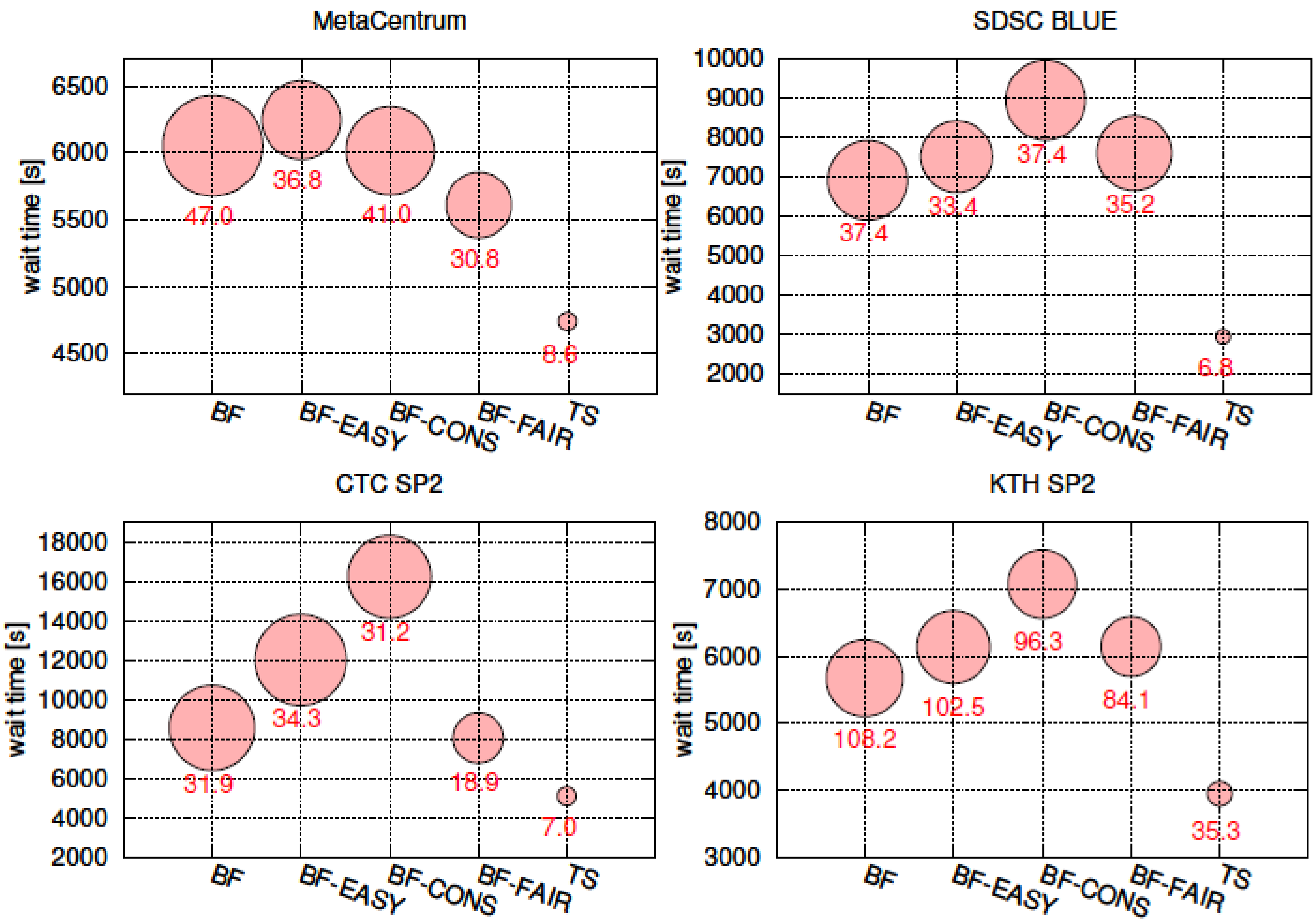
Experimental Results

- Alea simulator
  - Complex job scheduling simulator built on the top of optimized GridSim toolkit
    - Functionality (scheduling algorithms, visualization, ...)
    - Speed (optimized GridSim core)
- 6 data sets from Parallel Workloads Archive
  - MetaCentrum (806 CPUs, 103,656 jobs during 5 months)
  - KTH SP2 (100 CPUs, 28,489 jobs during 11 months)
  - CTC SP2 (338 CPUs, 77,222 jobs during 11 months)
  - SDSC SP2 (128 CPUs, 59,725 jobs during 24 months)
  - SDSC BLUE (1,152 CPUs, 243,314 jobs during 34 months)
  - HPC2N (240 CPUs, 202,876 jobs during 42 months)

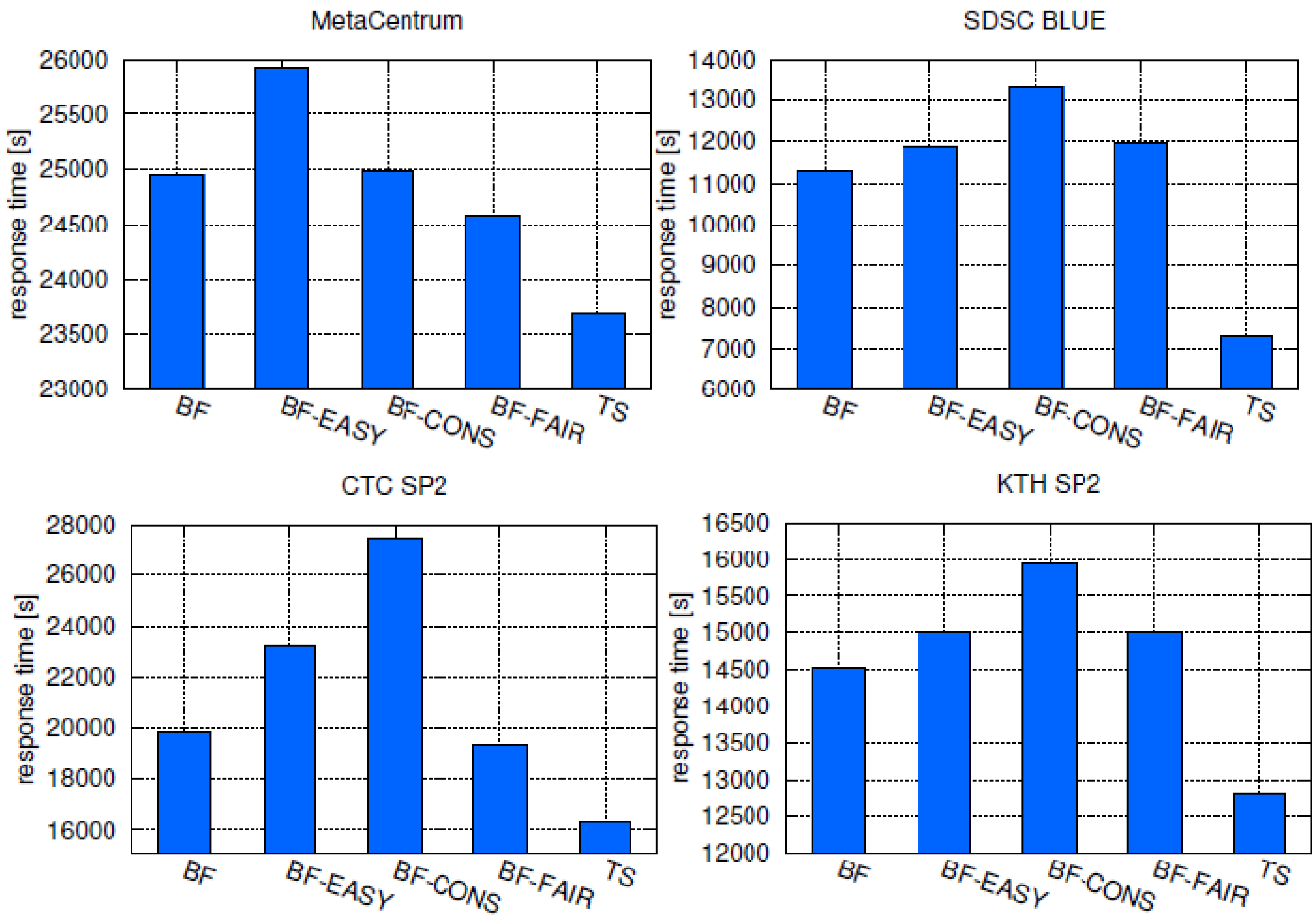
Algorithms

- Experimental evaluation of TS against
  - FCFS
    - Bad, offscale-high results
    - Not shown in the graphs
  - Backfilling without reservations (BF)
  - EASY backfilling (first job gets a reservation) (BF-EASY)
  - Conservative backfilling (every job get a reservation) (BF-CONS)
  - Backfilling without reservations + Fairshare (BF-FAIR)

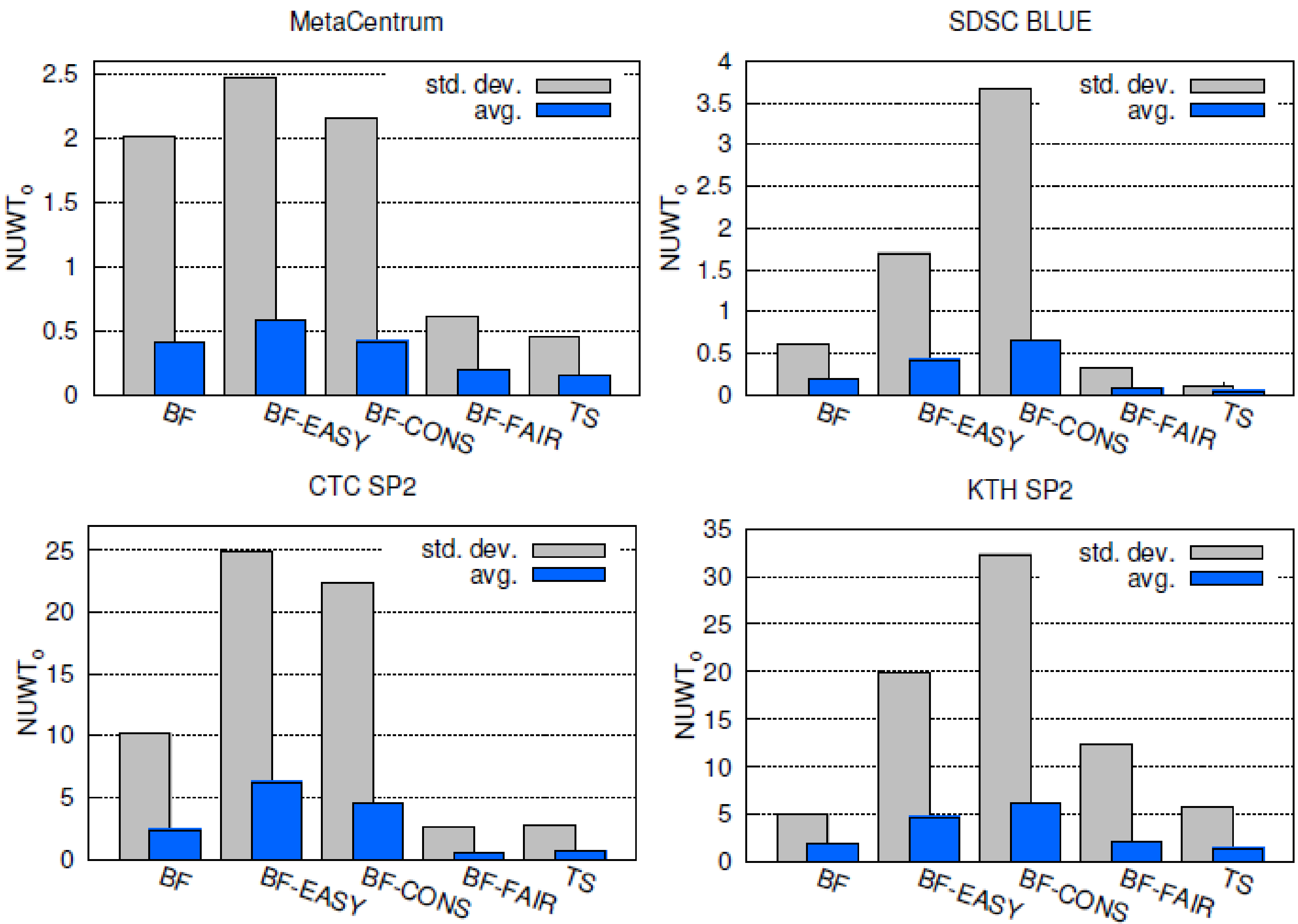
Slowdown + Wait time



Response time



Fairness

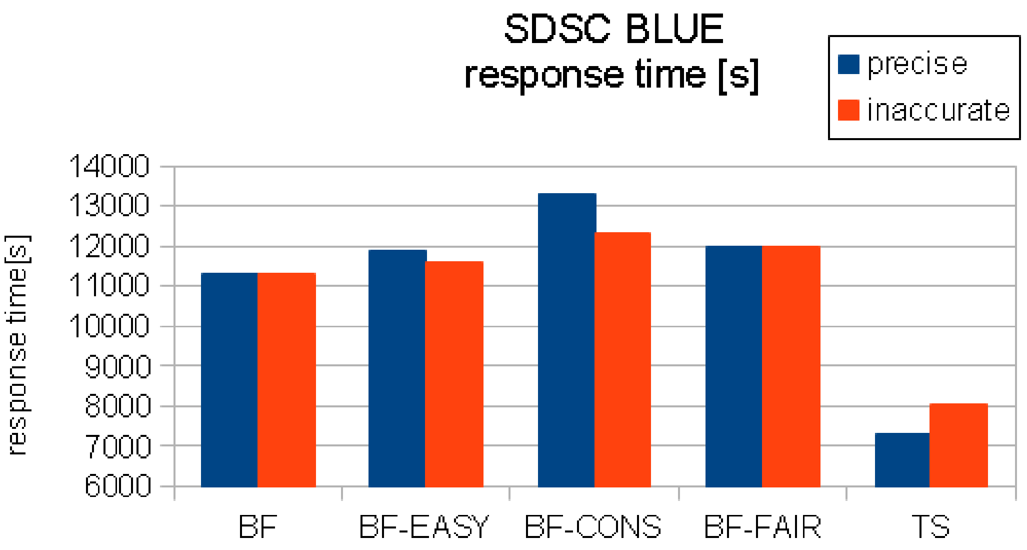


MetaCentrum
SDSC BLUE
CTC SP2
KTH SP2
std. dev.
avg.
$NUWT_o$
BF
BF-EASY
BF-CONS
BF-FAIR
TS

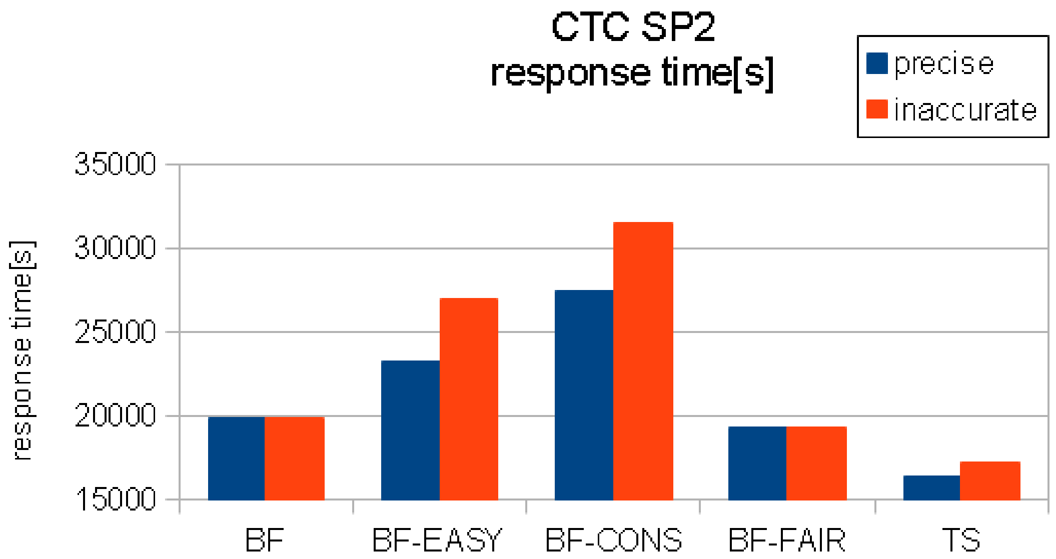
Conclusion

- Simple but powerful extension of Conservative backfilling
  - Evaluation and optimization
    - “Controlled” re-backfilling
- Significant improvement
  - Classical criteria
  - Fairness-related criteria
  - Time efficient
- Can be used when job runtime estimates are inaccurate
  - It is only backfilling...
    - Schedule compression is needed when job completes earlier
    - Evaluation is not precise – still improving solutions are found regularly

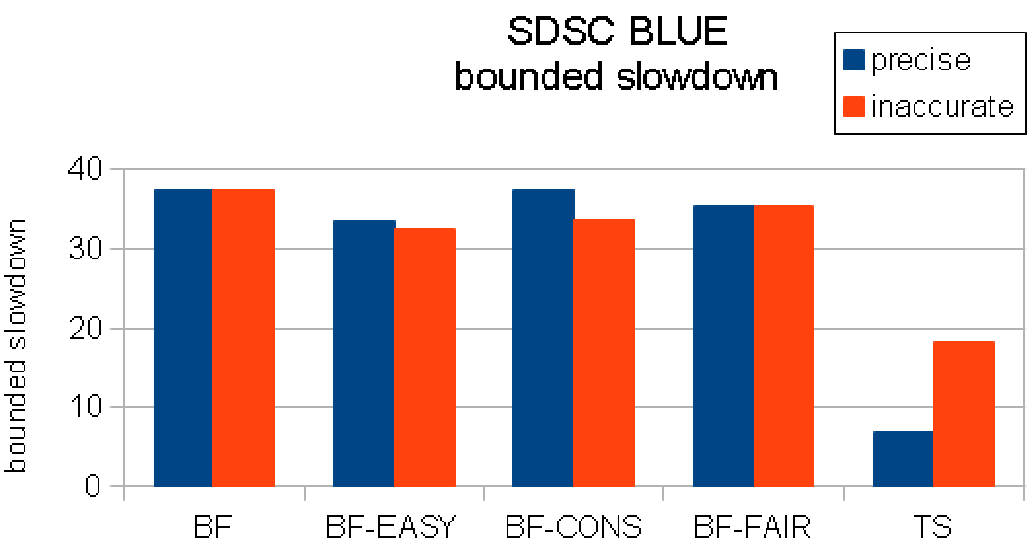
Example



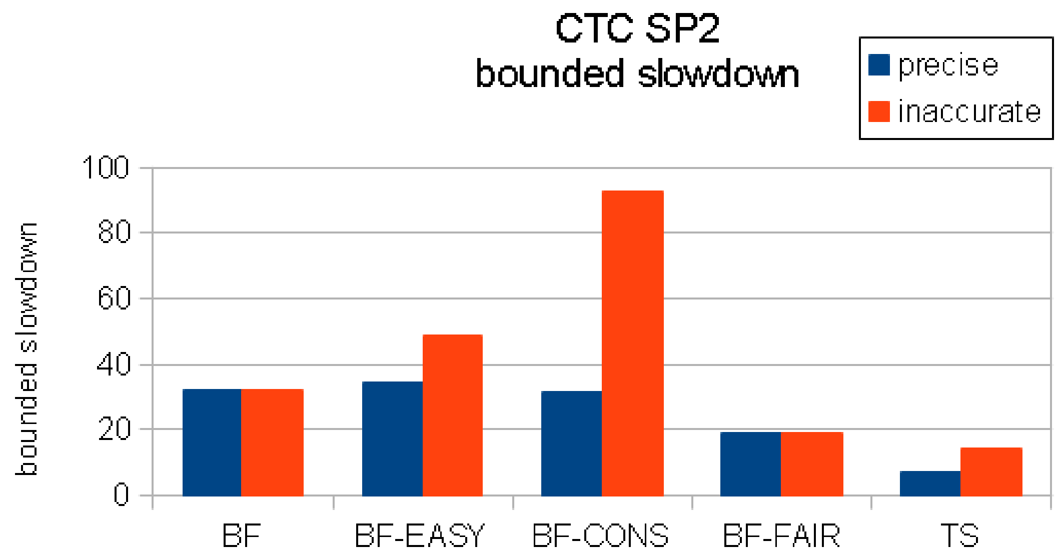
SDSC BLUE
response time [s]
precise
inaccurate
response time[s]
14000
13000
12000
11000
10000
9000
8000
7000
6000
BF
BF-EASY
BF-CONS
BF-FAIR
TS



CTC SP2
response time[s]
precise
inaccurate
response time[s]
35000
30000
25000
20000
15000
BF
BF-EASY
BF-CONS
BF-FAIR
TS



SDSC BLUE
bounded slowdown
precise
inaccurate
bounded slowdown
40
30
20
10
0
BF
BF-EASY
BF-CONS
BF-FAIR
TS



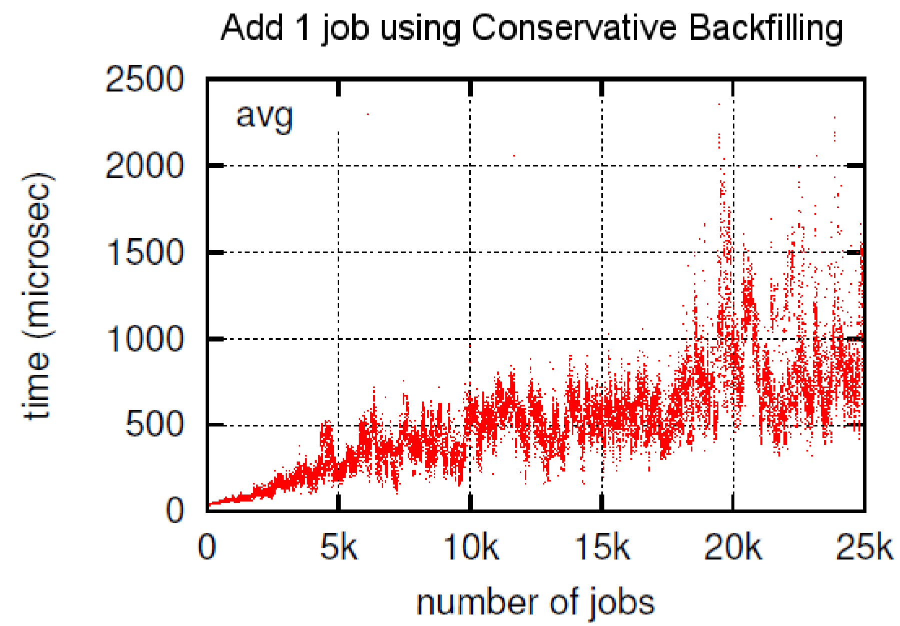
CTC SP2
bounded slowdown
precise
inaccurate
bounded slowdown
100
80
60
40
20
0
BF
BF-EASY
BF-CONS
BF-FAIR
TS

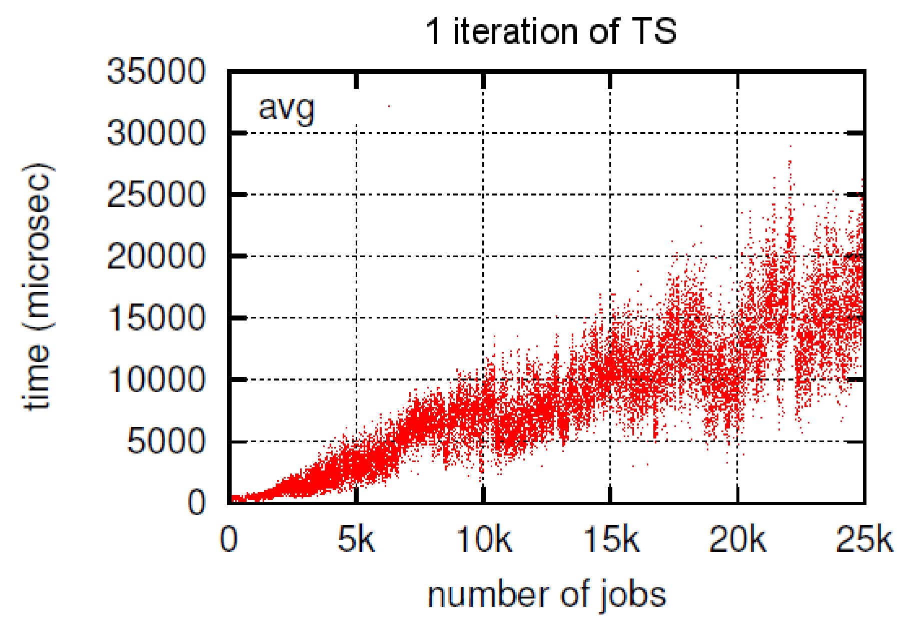
Ongoing Work

- Predictability
  - Conservative backfilling is predictable
  - Due to optimization the “reservations” are changed
    - Optimization delays some jobs wrt. initial assignment
- Multi-resource fairness
  - Memory, I/O
  - Berkeley's Dominant Resource Fairness - Fair Allocation of Multiple Resource Types
- Working implementation in TORQUE
  - First tests show better performance wrt. classical techniques
  - Further development toward practical usage

Runtime requirements

- Implementation in a real TORQUE scheduler
- Problem description:
  - 219 nodes with 1494 CPUs
  - Initial schedule consisting of 0..25,000 jobs
- Time needed to add 1 job
- Time needed to perform 1 iteration of TS





`qstat` command with prediction

vchlumsky@aeglos: ~

```
root@aeglos:~# qstat 3 -f
Job Id: 3.aeglos
    Job_Name = STDIN
    Job_Owner = pbstest@aeglos
    job_state = Q
    server = aeglos
    Checkpoint = u
    ctime = Mon Nov 28 16:43:46 2011
    qtime = Mon Nov 28 16:43:46 2011
    schtime = Mon Nov 28 16:47:20 2011
    schnode = node1
    euser = pbstest
    egroup = pbstest
    submit_args = -l nodes=1:ppn=4,mem=3gb,walltime=108

root@aeglos:~#
```

planned start time (schtime)

node name (schnode)