Fair Load Balance on Parallel Systems for QoS

Luis Orleans and Pedro Furtado

Univ. of Coimbra
CISUC
Portugal
The Context of this work

- ADAPT-DB project: middleware for backend servers with autonomic provision of QoS for requests
- Requests from multiple clients are posed against one or more servers
- There may be a heavy load

- BEST possible QoS qualities:
  - Best THROUGHPUT
  - Best RESPONSE TIME GUARANTEES
  - (Other QoS issues such as dependability)

- paper proposal is useful in other load-balancing environments as well
Problem formulation

• Typically, Load Balancing Algorithms for Parallel Systems have NO SPECIFIC CONCERNS FOR TIME CONSTRAINTS <deadlines>

• e.g. Least Work Remaining (LWR)

• Under Heavy load,
  – tasks may take more time than acceptable
  – big tasks are especially responsible for it

• What can we do about this?
What we show here

• Why traditional algorithms have this problem

• Propose algorithm:

ORBITA: On-demand Restriction for Big Tasks
  - Best possible THROUGHPUT
  - FAIR to all tasks
Basic Load Balancing

- several parallel servers compute incoming requests or tasks
- dispatched to them according to a load balance algorithm

• OBJECTIVE: balance load, execute all requests as fast as possible => Best Effort

• How much will a specific request take, is it acceptable? NO PROVISION
Under Heavy Load...

- High Request Arrival Rate...
  - All requests may take tens of times longer to execute than they would take in a less stressed server
  - BIG TASK: this time may be too much

- Best-Effort does not look at time guarantees
  - Tries to process everything as fast as possible
  - Some tasks may miss deadlines, especially BIG TASKS

- Admission Control?
  - with the highest possible THROUGHPUT
  - FAIR for any request duration
From Related Work

• Requests are usually heavy tailed
  • Small portion of tasks is responsible for half of the load (BIG TASKS)
  • Large amount of small tasks

• Is LWR the best possible algorithm?
  – There are works showing that LWR is optimal for exponential and unknown task sizes
  – When task sizes are known and heavy-tailed, algorithms discriminating tasks based on duration can produce better results than Least-Work-Remaining (LWR)
    • [Harchol-Balter et al. 1999], [Harchol-Balter et al. 2002]

• “The main issue involving LWR is the absence of control over big and small tasks mixing inside the same server”
Variability and Concurrent Executions

- Low variability workloads: all Load-Balancing algorithms have similar results
- High variability (typical): problems! Many tasks fail the deadline
  - Especially BIG TASKS -> UNFAIRNESS

Typical, heavy-tailed Task Distribution

- CE = number of requests being served simultaneously
- A CE that is acceptable for small tasks to end within the deadline might be unacceptable for large tasks!
Our Proposal: ORBITA

• ORBITA: On-demand Restriction for Big Tasks
• Separate small and big tasks
• Introduce Admission Control for BIG TASKS
  – Short tasks will always run immediately in short-task server(s)
    • Large number and large throughput
  – Big tasks will run in other server(s) and have their admission controled
    • Only accept big task if admission does not make it or other tasks miss their deadlines
    • Smaller number, guarantees that will not take large amounts of time
ORBITA

for each task that arrives:

if task is a small task

server := server_list -> first_server
send(task, server)

else

server := server_list -> second_server
biggest_task := biggest_running_task(server)
max_ce := LOWER_BOUND(deadline/biggest_task)
if number_of_running_tasks(server) >= max_ce
    NOT_ADMITT(task)
else
    send (task, server)
Experimental Setup

- Simulator
  - Number of servers (2 in this paper)
  - Tasks arrival rate (exponential distribution)
  - Task size distribution (Pareto distribution)
    (min size=0.001 sec, max size=10 secs)
  - Time constraints (deadline)
    (20 secs)
  - Load-Balancing algorithms tested
    (TAGS, SITA-E, LWR, ORBITA)
  - Minimum Size of "big tasks"
    (less than 1 sec is SMALL TASK)
    (more than 1 sec is BIG TASK)
Tested Algorithms

- ORBITA
- LWR
- TAGS (approach for unknown durations)
  - All tasks are dispatched to first server
  - If running for too long (big task), kill and restart it in second server
- SITA-E
  - Big tasks go to first server
  - Small tasks go to second server
  - No admission control
Pareto Generation

\[ f(x) = ak^a x^{-a-1}, a, k > 0, x > k \]

- For each Variance factor \( a \) (examples)
  - \( a = 0.1 \) (3.85% BIG tasks = some big tasks)
  - \( a = 0.3 \) (0.73% BIG tasks = very few big tasks)
Throughput with 3.85% BIG tasks

- SITA-E and ORBITA are better than LWR or TAGS
  - they do not mix small and fast requests into the same servers

- And next slides show the reason
  - Small tasks do not miss deadline in SITA-E and ORBITA even at 10 req/sec
  - With LWR or TAGS even small tasks fail deadlines sometimes
SMALL TASKS miss rate

- LWR and TAGS fail deadlines increasingly as arrival rate increases, even for SMALL TASKS!
BIG TASKS miss rate

- For big tasks SITA-E also fails the deadlines
- The miss rates are catastrophic for any of these
- ORBITA does not fail any deadline !! ADM. CTRL

L. Orleans, P. Furtado, ICPP 2007
Throughput with 0.73% BIG tasks

- Differences not significant, although LWR, ORBITA a bit better
  - Smaller number of big tasks, advantage is not obvious in this graph ...
  - ... but BIG TASKS always MISSED DEADLINES IN LWR and frequently in SITA-E
For SMALL TASKS

- **3.85% BIG tasks**: SITA-E and ORBITA have better throughput due to less misses
- **0.73% BIG tasks**: no major difference
**BIG TASKS: Throughput**

- **3.85% BIG tasks:**
  - Only ORBITA provides a significant throughput
    - For all other approaches big tasks almost always fail deadlines

- **0.73% BIG tasks**
  - ORBITA is the best for high arrival rates, otherwise LWR can have similar or better throughput results
    - LWR schedules optimally if we do not consider deadlines
    - There are too few very big tasks in this experiment, big tasks server is under-utilized
Conclusions

• Proposed ORBITA:
  - fast small tasks all run on a single server and never miss their deadlines
  - big tasks run on another server and have their admission controlled by the system

• Experiments showed that:
  - ORBITA handles High task durations variability and heavy load better than other algorithms
  - ORBITA is FAIRER than other algorithms, since tasks of all size intervals end well
Current Work

• Generalizing the algorithm
  – How to avoid server under-utilization with ORBITA?
  – Can we mix LWR and ORBITA to get benefits of both?
  – How to determine small and big tasks automatically and adaptively?
  – Generalize to multiple duration classes and any number of servers
• Thank You!

• eden.dei.uc.pt/∼pnf
• pnf@dei.uc.pt
• lorleans@dei.uc.pt
Conclusions

• Most load-balancing algorithms are not aware of response times and follow best-effort.

• Number of small tasks is much greater than the number of big tasks.

• Big tasks are the main responsible for making requests miss deadlines.

• UNFAIRNESS: BIG tasks fail more deadlines.
Why time QoS is relevant

- Under heavy load there are two options:
  - Degrade response for everyone indefinitely (how much is enough)?
  - Apply Admission Control
- Depending on request/task type, it makes sense to have an upper bound on expected response time:
  - E.g. an interactive request should end within 15 seconds at most
  - WHY? Because it makes no sense to have the submitter waiting for longer than he perceives to be acceptable
- Given time constraints, bigger tasks are more under stress than small ones
Observations

• Tasks are frequently heavy-tailed
  • Very small portion of tasks is responsible for half of the load (BIG TASKS)
  • Large amount of small tasks
  • Pareto-like:

![Graph showing task distribution over time intervals](chart.png)
**Concurrent Executions (CE)**

- Number of Concurrent Executions (CE) in the system is crucial when there are deadlines
  
larger CE => larger probability of missing deadlines
  
Especially when there are BIG TASKS

![Response Times Graph](image)

Time (msecs)

**CE** – Number of Requests Being Served Simultaneously
Related Work

• Load-Balancing
  – Typically maximize throughput or minimize mean response time
  – When task sizes are known and heavy-tailed, SITA-E produces better results than Least-Work-Remaining (LWR)
  – There are works showing that LWR is optimal for exponential and unknown task sizes

• QoS
  – Number of simultaneous executions, admission control, models, etc..

• Real-Time scheduling ans task assignment