Resource Management Part III

Alex Barapp, Spring 2015

Challenges in Modern Data Centers Management, Spring 2015
Information provided in these slides is for educational purposes only.
Outline

• Single-cluster scheduling (reminder)
  • Proportional-share and resource matching (allocation)

• **Meta-scheduling**
  • Designing a global solution
    • Balancing load among the sites
  • Overall architecture
    Job lifecycle demonstration
  • Ensuring shares globally
    • Three examples

• Practical challenges in building a batch system
  • Scalability, robustness, usability
Reminder – Single Cluster
Reminder: Proportional-share scheduling

• **Target tree** defines what each node “should-get”

• **Measured (usage) tree** defines what each node is “getting-now”
  - Assuming no history considered

• **Fair-share (priority) tree** defines the order jobs are selected for execution
  - Translates the hierarchical “tree” structure into a **flat list** of a jobs
Reminder: Resource matching (allocation)

• Receives jobs to execute from the scheduling phase (proportional share)
  • Jobs received one at a time or several jobs together (look-ahead)
• Matches the jobs’ requirements with available resources
  • Best-fit, Worst-fit, etc.

![Diagram of resource matching](attachment://image.png)
Reminder: Single-cluster scheduling

1. Job Scheduling
Select next job(s) to execute

(2) Flat list (queue)
These are the selected jobs

3. Resource Matching
Select where to execute the job(s)
Meta-Scheduling
Global environment

1. Multiple data centers (pools of resources)
   - Each managed by its own RM system (scheduler)
     - Locally managed configuration
   - Need to be combined into a **single shared** pool of resources
     - Load balancing (improve global utilization)
     - Simplified management

2. Users located in multiple sites
   - But want to utilize remote servers
   - Working on cross-site projects
Global environment – Google (example)

Google’s official information

Challenges in Modern Data Centers Management, Spring 2015
Designing a global RM solution

• Requirements
  1. Must be scalable
  2. Take into consideration the different setup at each site
  3. Provide “locality” feeling for the submitting user (fast response, etc.)
  4. Prevent multiple execution of same job (when not allowed)

• In the following examples assume we have 3 data centers
  1. SC – Santa Clara
  2. FM – Folsom
  3. HA - Haifa
Designing the solution (1)

• One central global scheduler
  • Jobs enter through a single entry point
  • All servers (resources) known to the scheduler

• Cons – Not scalable, single-point-of-failure, clusters have different setup
• Pros – Simple, theoretically may lead to optimal utilization
Designing the solution (2)

- Single-level / multiple schedulers
  - Jobs enter through different entry points (G’s)
  - All servers are known to all schedulers

- Cons: Race-conditions, clusters have different setup, etc.
Designing the solution (3)

• Hierarchical (2+) layers of schedulers
  • Jobs enter at the top level (different entry points)
  • Top level distributes the load (meta-scheduler)
  • Bottom level manages the servers (locally, per-site)
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Solution #1</th>
<th>Solution #2</th>
<th>Solution #3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td>Poor</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Single point of failure</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Scheduler Types</td>
<td>Global</td>
<td>Global</td>
<td>Global + local</td>
</tr>
<tr>
<td>Number of components</td>
<td>1</td>
<td># of sites</td>
<td># of sites * 2</td>
</tr>
<tr>
<td>“Locality feeling” for users</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Dealing with non-standard local setup</td>
<td>Hard Need to manage all servers in global scheduler</td>
<td>Very hard Need to manage all servers in many global schedulers</td>
<td>Local + migration rules</td>
</tr>
</tbody>
</table>

Challenges in Modern Data Centers Management, Spring 2015
Selected solution: #3

- Users at different sites submit and query jobs at their local entry points (G)
  - User gets locality feeling
- Jobs are forwarded to local RMs (at the local or remote site)
  - How to balance the load? next slide
- Each local RM independently manages its cluster
  - Resource matching, etc.
- Pros: Scalable, no global single point of failure
Balancing the load among the sites

1. Information-based (dynamic info)
   • We know the current load conditions at the different sites

2. Information-less (static info)
   • We don’t know enough about the load
   • We don’t know anything about the load
1. Information-based load balancing

• Dispatch job to least-loaded site (using e.g., avg. wait time at the local site)
  • If job not running after X seconds, dispatch to second least-loaded, etc.

• Pros 😊
  • Deterministic approach, easy to explain and debug

• Cons 😞
  • Stability issues (may result in everyone sending to same site)
  • May require lots of information about the remote sites
  • “Load” metrics may not be accurate (wait time different for different jobs; wait time unknown for new jobs; site capacity is not well defined)
2. Information-less load balancing

- Send jobs to **all sites** with preference to local
  - While ensuring single execution (next slide...)

**Pros 😊**
- Data locality
- Opportunistic: will get slot (start executing) in the first available site

**Cons 😞**
- More difficult implementation
- More difficult to debug
2. Information-less: Ensuring single execution

• Job sent to “G” (for example, G at HA site)
• “G” dispatches the job to more than one “L”s (for example, HA, SC, FM)
  • How to ensure that the job will execute only once?
2. Information-less: ensuring single execution

1. Job is submitted to a single entry point (“G”)
   • “G” is responsible for that job (“home-node” semantics)

2. “G” dispatches the job to multiple sites (multiple “L”s)

3. When a local RM (“L”) matches the job with a server – it asks for “approval” from “G”

4. Only single execution will be approved by “G”
   • Subsequent requests will be denied

5. This “handshake” costs overhead: few seconds delay
   • Thus, latency-critical jobs should be sent directly to site layer
Overall architecture
1. Job submitted by user in site A
2. Job is prioritized based on the allocation trees (proportional share)
3. Job is added to dispatch queue
4. Job is dispatched to all sites ("dispatch everywhere")
5. Job finds resource in site B
6. The job verifies it can run in site B
7. The job starts to run on a host
8. Job is marked as running for the user (usage tree updated accordingly)
1. The job finishes on a host

2. Event is reported locally to L

3. Event is reported to G

Overall architecture – Job lifecycle

Challenges in Modern Data Centers Management, Spring 2015
Additional considerations for global environment

• Are data centers owned by same or different organization?
  • Internal data centers vs. external cloud (Amazon)
  • Security model (authentication, authorization) of who/what/where
    • For internal, security is less of an issue.
    • Global authentication and authorization mechanism used cross-site

• Do the cluster components (local and global) belong to same family?
  • Possibly developed within organization
Ensuring shares globally
Problem statement

• Proportional share algorithm works great in a single site
  • Single target tree and measured (usage) tree
  • Selecting the next job to execute is easy (ensuring the shares)

• With Meta-scheduling we have multiple trees
  • Each “G” maintain it own trees (target, usage, etc.)

• How to configure the trees (target trees) to ensure shares globally?
  • Across all sites...
2 projects P1, P2, with users in two sites

2 sites, SA, SB with 10 servers each

Shares we want to enforce globally (in both sites combined)
How to globally ensure shares? Example #1

P2: j1, j3, j5
P1: j2, j4

P2: j6, j8, j0
P1: j7, j9

Site A

Site B
How to globally ensure shares? Example #1

• Both SA and SB have 10 servers each
• Both P1 and P2 have users active in both sites

Requirement: P1 and P2 should achieve global share of 40% / 60%

<table>
<thead>
<tr>
<th></th>
<th>SA</th>
<th>SB</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>P2</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Solution:  Define the same share (40/60) in both target trees (in both sites)
Since both projects are constantly submitting jobs in both sites 
the global share will also be 40/60
How to globally ensure shares? Example #2

P2: j1, j3, j5
P1: j2, j4, j11, j12

Site A

Site B

No active users

Challenges in Modern Data Centers Management, Spring 2015
How to globally ensure shares? Example #2

- Both SA and SB have 10 servers each
- P2 has active users only in SA, P1 has in both sites
Requirement: P1 and P2 should achieve global share of 40% / 60%

<table>
<thead>
<tr>
<th></th>
<th>SA</th>
<th>SB</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>P2</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Solution direction: in SA, give much higher share to P2 (since P1 can get it in SB)
- Static (one-time configuration) solution will work 😊
How to globally ensure shares? Example #3

P2: j1, j3, j5
P1: j2, j4

P2: j6, j8, j0
P1: j7, j9

P2 is not active on weekends
How to globally ensure shares? Example #3

- Both SA and SB have 10 servers each
- Both projects have users active in both sites
  - But P2 is not active on weekends in SB

Requirement: P1 and P2 should achieve global share of 40% / 60%

<table>
<thead>
<tr>
<th></th>
<th>SA</th>
<th>SB</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>P2</td>
<td>Yes</td>
<td>Sometimes</td>
</tr>
</tbody>
</table>

Solution direction: in SA, give much higher share to P2 (since P1 can get it in SB) – but only when it is not active in SB
- Static solution will not work 😞
Solution #1: manual at **global** level

- Desired configuration \((P1=40, P2=60)\) set on the top level in all global schedulers’ configuration
- Manual share configuration at the low level to adjust to desired global share

**Cons** – manual calculations, coordination with support team in different sites
Solution #2: manual at local level

- Desired configuration \((P1=40, P2=60)\) set on the low level in all global schedulers’ configuration
- Manual share configuration at the global level to adjust to desired global share

- **Cons** – manual calculations, coordination with support team in different sites
Solution #3: automated, site level

• Desired configuration (P1=40, P2=60) set in the high-level policies management component (“RA”)

• Automatically controlling top level allocations to reach desired global share

• Cons – complex development, additional “player”

• Pros – automated system
Summary: RM parts I-III

• Problem is collection of intermixed constraints
  1. Guarantee the shares
     • While minimizing fragmentation
     • While avoiding starvation
  2. With or without information e.g., load information
  3. Where servers may be non-uniform
  4. And jobs may have different requirements
  5. Across multiple data centers...
Summary: RM parts I-III, cont.

• Amount of information available affects the solution
  • Job runtime in the case of reservation (RM part II)
  • Load and resources availability in the case of meta-scheduling (RM part III)

• Information is not always available
  • e.g., job runtime is very hard to predict (Predictive analytics – lecture 9)
Practical challenges in building a batch system
Practical challenges

• Scalability
  • Number and characteristics of jobs
  • Number of resources
  • Number of simultaneous queries

• Robustness
  • Quality of service
  • Malicious users
  • Unstable resources environment
  • Unstable jobs

• Usability
  • For users
  • For administration
Scalability

• Define, test and document maximum cluster capacity
• Use multi-threaded approach
• Meta-data synchronization:
  • Define static and dynamic data
  • Send only dynamic data on change
  • Define asynchronous protocols
Robustness

• Define different QoS for different users
• Block attacking users
• Define limits on users/groups/projects.
• Define limits on hung/runaway jobs:
  • Time, resource consumption (too low or too high)
• Provide built-in monitoring of cluster components
Usability

- For users:
  - Ask only for necessary parameters from the users, define defaults for others.

- For admins:
  - Enforce standards through the product.
  - Built-in monitoring tools
  - For developers
  - Built-in debugging tools
  - Logging of necessary information for troubleshooting