IN4391 Distributed Computing Systems

Scheduling in Distributed Computing Systems

Alexandru Iosup
Parallel and Distributed Systems Group
Delft University of Technology
(From the previous lecture)

The Client/Server Model

- Client-server model = message exchange with specific pattern: the client sends requests, then the server replies
- Special forms of client-server model:
  - in imperative languages, **remote procedure calls (RPC)**
  - in OO languages, **remote method invocations (RMI)**
  - on the web, **http**
Design of Distributed Systems: A General Approach (Intermezzo)

- Understand Problem
  - Workload
  - Participants

- Design mechanism
  - Design basic mechanism
  - Explore properties of basic mechanism
  - Extend mechanism

- Iterate
  - Example, only as flavor

- Use mechanism in larger constructs:
  - **Organization** = mechanisms, methods, algorithms, etc.
  - **Architecture** = how do the elements of organization interact
  - **Top-to-bottom approach** = architecture then organization

The actual lecture
**Sync Remote Procedure Call**

Simple mechanisms can enable entire markets

- Wait for result
- Request
- Reply

**Async Remote Procedure Call**

Simple mechanisms can lead to big improvements

- Wait for accept
- Request
- Result
- ACK

---

IN4391 Distributed Computing Systems
Distributed Workload Management

- **Main goal**: establish a **mutual agreement** between resource providers and resource consumers

- **Resource Consumers**
  - Use programming model for creating jobs
  - Selection and aggregation of resources

- **Resource Providers**
  - Provide resources for executing user jobs
  - Maximize resource utilization or Maximize revenue (= income - penalties)

- *(Workloads, Mechanisms + Architecture + Organization follow)*
Agenda

1. Remote Procedure Calls, Mechanisms, Complete Systems
2. Workloads in Distributed Systems
   1. An Overview of Job Types
   2. Parallel Jobs in Batch Production Environments
   3. Bags of Tasks in Grids
3. Architectures for Scheduling in Distributed Systems
4. Scheduling in Distributed Systems
5. Conclusion
Job Types in Distributed Systems

- **Parallel jobs**
  - Extension of high-performance computing

- **Sequential jobs/ bags-of-tasks**
  - E.g., parameter sweep applications (PSAs)
  - These account for the vast majority of jobs of grids
  - Leads to high-throughput computing

- **Workflows**
  - Multi-stage data filtering, etc
  - Represented by directed (acyclic) graphs
  - MapReduce applications

- **Miscellaneous**
  - Interactive simulations
  - Data-intensive applications
  - Web applications, online gaming, etc.
Workloads in Batch Production Systems

Q: What are the typical sizes of jobs in Batch Production Systems?

Q: Typical utilization of Batch Production Systems?

Q: What to do about selected users?

What is a Bag of Tasks (BoT)?

**BoT** = set of jobs sent by a user...

\[ W_u = \{ J_i | user(J_i) = u \} \]

...at most \( \Delta \) s after the first job

\[ ST(J') \leq ST(J) + \Delta \]

- Why Bag of *Tasks*? From the perspective of the user, jobs in set are just tasks of a larger job.
- A single useful result from the complete BoT.
- Result can be combination of all tasks, or a selection of the results of most or even a single task.

Applications of the BoT Programming Model

• Parameter sweeps
  • Comprehensive, possibly exhaustive investigation of a model
  • Very useful in engineering and simulation-based science

• Monte Carlo simulations
  • Simulation with random elements: fixed time yet limited inaccuracy
  • Very useful in engineering and simulation-based science

• Many other types of batch processing
  • Periodic computation, Cycle scavenging
  • Very useful to automate operations and reduce waste
BoTs = Dominant Programming Model for Grid Computing

From Jobs [%]

From CPUTime [%]

Iosup and Epema: Grid Computing Workloads.

Agenda

1. Remote Procedure Calls, Mechanisms, Complete Systems
2. Workloads in Distributed Systems
3. **Architectures for Scheduling in Distributed Systems**
   1. Single-Cluster Architectures
   2. Multi-Cluster Architectures
4. Scheduling in Distributed Systems
5. Conclusion
Architectures for Distributed Workload Management

- **Goals**
  - Execute entire workloads
  - Ensure Service Level Agreements/Objectives (SLAs/SLOs)

- **Architectural issues**
  - (user) Resource selection
  - (system) Complexity in managing the architecture
  - Load imbalance = Imbalance in the use of machines
  - Central points = Who owns the central point? Who pays for it? Failures?
  - Ability to scale to large number of nodes/cores (2013 target: 1M cores)

Q: Why is load imbalance important?

Q: Why is central point ownership important?

“All jobs finish within 1d and 99% jobs within 1h and cost<$10k”
Single Cluster Architecture
Centralized Scheduler

Headnode

Node

Node

Node

Job

Now for some easy questions

Q: Is res. selection easy?

Q: Is this system complex?

Q: What happens with load imbalance in this system?

Q: Any central point?

Q: Can this system scale?
Single Cluster Architecture Distributed (Condor)

Remote execution

Basic operation of Condor:

1a jobs send classads to the matchmaker

1b machines send classads to the matchmaker

1c matchmaker matches jobs and machines

1d and notifies the submission machine

2a which starts a shadow process that represents the remote job on the execution machine

2b/c and contacts the execution machine

3b/c on the execution machine, the actual remote user job is started

Q: Why is this architecture distributed?

Q: What is the benefit of remote execution here?
Agenda

1. Remote Procedure Calls, Mechanisms, Complete Systems
2. Workloads in Distributed Systems
3. **Architectures for Scheduling in Distributed Systems**
   1. Single-Cluster Architectures
   2. Multi-Cluster Architectures
4. Scheduling in Distributed Systems
5. Conclusion
Multi-Cluster Architectures
Independent Clusters

Load imbalance?

Resource selection?

<table>
<thead>
<tr>
<th>Site-A</th>
<th>Site-B</th>
<th>Site-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>1k</td>
<td>0.5k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Site-D</th>
<th>Site- E</th>
<th>Site-F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5k</td>
<td>-</td>
<td>0.5k</td>
</tr>
</tbody>
</table>

Number of nodes
Number of local users

Site-G
- 20
Load Imbalance in Independent Clusters

- **Overall workload imbalance**: normalized daily load (5:1)
- **Temporary workload imbalance**: hourly load (1000:1)
Multi-Cluster Architectures
Centralized Meta-Scheduler

Legend:
- Cluster name
- # users
- # resources
- Job flow

Site-A
- 10
- 10

Site-B
- 200
- 1k

Site-G
- -
- 20

Scale?
Failures?
Root ownership?
Multi-Cluster Architectures

Hierarchical Meta-Scheduler

Scale?
Failures?
Root ownership?
Multi-Cluster Architectures
Fully Decentralized/Load-Sharing
Multi-Cluster Architectures

Fully Decentralized/Load-Sharing (Alternative 1)

- Protocol **between Central Managers:**

  ![Diagram showing communication between Central Managers and other processes]

  **Disadvantage:**
  - the matchmaker has to be modified
Multi-Cluster Architectures

Fully Decentralized/Load-Sharing (Alternative 2)

- GateWays (networking)

Advantages:

- transparent to the matchmaker
- no component to be maintained by a third party
- in the GWs, any access policies and resource sharing policies can be implemented

**Conclusions:**

- Design considerations for Condor Flocking are still very valid when joining systems.
- Nice, clear, transparent research solution that was too complex in practice.

Q: What is the complexity of this approach?
The Delegated MatchMaking Architecture

Q: What is the scalability of this approach?

Delegated MatchMaking Architecture = Hybrid hierarchical/ decentralized architecture for grid inter-operation

Multi-Cluster Architectures: A Hybrid Architecture [2/3]

The Delegated MatchMaking Mechanism

Q: What is the complexity of this approach?

The Delegated MatchMaking Mechanism=
Delegate Resource Usage Rights,
Do Not Delegate Jobs

Alexandru Iosup, Dick H. J. Epema, Todd Tannenbaum, Matthew Farrellee, Miron Livny:
Multi-Cluster Architectures: A Hybrid Architecture [3/3]

Potential Gain of Grid Inter-Operation

Delegated MatchMaking vs. Others

(Higher is better)

- DMM
  - High goodput
  - Reasonable overhead
  - [see thesis]

Delegated MatchMaking delivers good performance

- DMM
- Decentralized
- Centralized
- Independent

Alexandru Iosup, Dick H. J. Epema, Todd Tannenbaum, Matthew Farrellee, Miron Livny:
Agenda

1. Remote Procedure Calls, Mechanisms, Complete Systems
2. Workloads in Distributed Systems
3. Architectures for Scheduling in Distributed Systems
4. **Scheduling in Distributed Systems**
   1. User-level scheduling
   2. Data center scheduling
5. Conclusion
Heuristics-Based Provisioning and Allocation Policies

- Provisioning
- Allocation

<table>
<thead>
<tr>
<th>Policy</th>
<th>Class</th>
<th>Trigger</th>
<th>Adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Startup</td>
<td>Static</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>OnDemand</td>
<td>Dynamic</td>
<td>QueueSize</td>
<td>No</td>
</tr>
<tr>
<td>ExecTime</td>
<td>Dynamic</td>
<td>Exec.Time</td>
<td>Yes</td>
</tr>
<tr>
<td>ExecAvg</td>
<td>Dynamic</td>
<td>Exec.Time</td>
<td>Yes</td>
</tr>
<tr>
<td>ExecKN</td>
<td>Dynamic</td>
<td>Exec.Time</td>
<td>Yes</td>
</tr>
<tr>
<td>QueueWait</td>
<td>Dynamic</td>
<td>Wait Time</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Policy</th>
<th>Queue-based</th>
<th>Known job durations</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCFS</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>FCFS-NW</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SJF</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- Also looked at combined Provisioning + Allocation policies

The SkyMark Tool for IaaS Cloud Benchmarking
User-Level Scheduling

Experimental Tool: SkyMark

Provisioning and Allocation policies steps 6+9, and 8, respectively

User-Level Scheduling

Experimental Setup (1)

• Environments
  • DAS4, Florida International University (FIU)
  • Amazon EC2

• Workloads
  • Bottleneck
  • Arrival pattern

<table>
<thead>
<tr>
<th>Workload Unit</th>
<th>CPU</th>
<th>Memory</th>
<th>I/O</th>
<th>Appears in</th>
</tr>
</thead>
<tbody>
<tr>
<td>WU1</td>
<td>X</td>
<td></td>
<td></td>
<td>WL1</td>
</tr>
<tr>
<td>WU2</td>
<td></td>
<td></td>
<td>X</td>
<td>WL2, WL4</td>
</tr>
<tr>
<td>WU3</td>
<td></td>
<td></td>
<td>X</td>
<td>WL3, WL4</td>
</tr>
</tbody>
</table>

![CPU Load Graphs]

User-Level Scheduling

Experimental Setup (2)

- **Performance Metrics**
  - Traditional: Makespan, Job Slowdown
  - Workload Speedup One (SU1)
  - Workload Slowdown Infinite (SUinf)

\[
SU_1(W) = \frac{MS(W)}{\sum_{i \in W} t_R(i)}
\]
\[
SU_{\infty}(W) = \frac{MS(W)}{\max_{i \in W} t_R(i)}
\]

- **Cost Metrics**
  - Actual Cost (Ca)
  - Charged Cost (Cc)

\[
Ca(W) = \sum_{i \in \text{leased VMs}} t_{stop}(i) - t_{start}(i)
\]
\[
Cc(W) = \sum_{i \in \text{leased VMs}} [t_{stop}(i) - t_{start}(i)]
\]

- **Compound Metrics**
  - Cost Efficiency (Ceef)
  - Utility

\[
Ceef(W) = \frac{Cc(W)}{Ca(W)}
\]
\[
U(W) = \frac{SU_1(W)}{Cc(W)}
\]
User-Level Scheduling
Performance Metrics

- Makespan very similar
- Very different job slowdown

User-Level Scheduling
Cost Metrics

Charged Cost ($C_c$)

February 28, 2013
User-Level Scheduling Cost Metrics

Actual Cost

Charged Cost

- Very different results between actual and charged
  - Cloud charging function an important selection criterion
- All policies better than Startup in actual cost
- Policies much better/worse than Startup in charged cost

User-Level Scheduling
Compound Metrics

• Trade-off Utility-Cost still needs investigation
• Performance or Cost, not both: the policies we have studied improve one, but not both


No single policy good for all cases

Q: Can we achieve optimal scheduling?
User-Level Scheduling Optimization: The ExPERT Scheduler

#machines > #unfinished tasks

![Graph showing the relationship between the number of remaining tasks and time, with a distinction between throughput and tail phases.]

User-Level Scheduling Optimization: The ExPERT Scheduler

- $D$ - instance deadline, $T$ - replication time
- Reliable machine used to ensure task completion
- $N$ tail instances at most on unreliable resources
- $M_r$ - max ratio of reliable to unreliable resources

User-Level Scheduling Optimization: The ExPERT Scheduler

ExPERT recommended:

\( N = 3, T = T_{ur}, D = 2T_{ur}, M_r = 0.02 \), in words:

Send \( N = 3 \) instances to the unreliable pool during the tail phase, each timed out after twice the average task time \( (D = 2T_{ur}) \). Send the next instance after the average task time passes \( (T = T_{ur}) \). Use only one \( (\#ur = 50, 50 \times M_r = 1) \) reliable machine at a time.

O. Agmon Ben-Yehuda, A. Schuster, A. Sharov, M. Silberstein, and A. Iosup,
ExPERT: Pareto-Efficient Task Replication on Grids and a Cloud,
IPDPS' 12.
User-Level Scheduling
Optimization: The ExPERT Scheduler

ExPERT can lead to optimality*

*Pareto frontier for optimal cost, approximated in online scheduling

- B: budget-based
- C: combine resources, no replication (N=1, T=0)
- C7: combine resources, replicate for tail (N=0, T=0)
- TR: all tail to reliable (N=0, T=D)
- TRR: all tail to reliable, rapidly (N=0, T=0)
- AUR: all to unreliable, no replication

Agenda

1. Remote Procedure Calls, Mechanisms, Complete Systems
2. Workloads in Distributed Systems
3. Architectures for Scheduling in Distributed Systems
4. **Scheduling in Distributed Systems**
   1. User-level scheduling
   2. **Data center scheduling**
5. Conclusion
Data Center Scheduling
Portfolio Scheduling [1/6]

• Intuition
  • No single policy is best for all cases

• The core of portfolio scheduling
  • Use a portfolio of policies
  • Only one policy selected at any one time
  • Automated selection of policies

Data Center Scheduling
Portfolio Scheduling [2/6]

- Portfolio = Provisioning x Allocation policies (or selection thereof)

Q: Which policies can be used with this approach?

- Policy selection
  - Interval between selections (=20s)
  - Via simulation

Q: Can all simulations finish?

Q: How to select a policy?
Data Center Scheduling
Portfolio Scheduling [3/6]

- Experimented with various workloads

Data Center Scheduling
Portfolio Scheduling [4/6]

Portfolio has good performance and cost

Portfolio delivers good utility

Data Center Scheduling
Portfolio Scheduling [6/6]

- Workload vs Policy change

**Best policies change over time**

Q: Is portfolio scheduling **useful**?

Agenda

1. Remote Procedure Calls, Mechanisms, Complete Systems
2. Workloads in Distributed Systems
3. Architectures for Scheduling in Distributed Systems
4. Scheduling in Distributed Systems
5. Conclusion
Conclusion Take-Home Message

From RPC to Distributed Workload Management = workload characteristics + mechanisms + architecture + scheduling

• Architectures
  • Single-cluster: centralized, distributed
  • Multi-cluster: centralized, (multi-level) hierarchical, distributed, hybrid

• Scheduling
  • User-level: various provisioning and allocation policies, online semi-optimal
  • Data center level: portfolio scheduling

• Reality Check: architectures and schedulers focus of Google, ...

http://www.flickr.com/photos/dimitrisotiropoulos/4204766418/
Thank you!
Suggestions? Questions?

- http://www.st.ewi.tudelft.nl/~iosup/research.html
- http://www.st.ewi.tudelft.nl/~iosup/research_cloud.html
- http://www.pds.ewi.tudelft.nl/

Alexandru Iosup

A.Iosup@tudelft.nl
http://www.pds.ewi.tudelft.nl/~iosup/ (or google “iosup”)
Parallel and Distributed Systems Group
Delft University of Technology