CloudBATCH: A Batch Job Queuing System on Clouds with Hadoop and HBase

Chen Zhang
Hans De Sterck
University of Waterloo
Outline

• Introduction
  • Motivation
  • Related Work
• System Design
• Future Work
Introduction - Motivation

- Users have hybrid computing needs
  - Legacy programs dispatched by traditional cluster systems
  - MapReduce programs requiring Hadoop
- Hadoop is incompatible with traditional cluster management systems
  - Hadoop is yet another system to manage cluster machines
  - Administrative barriers for deploying dedicated Hadoop cluster on existing cluster machines
  - Using Hadoop on public clouds (Amazon) is not cost-effective for large scale data processing tasks
- CloudBATCH makes it possible to
  - Use Hadoop to manage clusters for both legacy and MapReduce computing needs
Introduction – CloudBATCH

- CloudBATCH is built on top of Hadoop (job execution) and HBase (META-data management) with a scalable architecture where there is no single point of failure
- What CloudBATCH can do
  - Batch job queuing for legacy applications besides MapReduce jobs
  - Job queue management with individually associated policies, user access control, job priority, pre-scheduled jobs, etc.
  - Transparently handle jobs with simple non-NFS file staging needs
  - Task level fault tolerance against machine failures
  - Work complementarily with Hadoop schedulers, such as setting the minimum number of node allocation guaranteed to each queue
  - The HBase table records can also serve as a good basis for future data provenance supports
- What CloudBATCH cannot do
  - MPI-type jobs, reserve actual compute nodes, etc.
Introduction - Related Work (1)

- **Hadoop Schedulers**
  - Focus on optimizing task distribution and execution
  - Do not provide rich functionality for cluster level management, such as user access control and accounting, and advanced job and job queue management

- **Hadoop On Demand (HOD)**
  - Creates a Hadoop cluster on-the-fly by running Hadoop daemons through the cluster resource management system on a set of reserved compute nodes
  - Does not exploit data locality for MapReduce jobs because the nodes where the Hadoop cluster is created on may not host the needed data at all

- **Sun Grid Engine (SGE) Hadoop Integration**
Sun Grid Engine (SGE) Hadoop Integration

- Claims to be the first cluster resource management system with Hadoop integration
- Creates a Hadoop cluster on-the-fly like HOD with better data locality concerns
- Potential risks of overloading a compute node as a result of catering for data locality concerns (non-exclusive node usage)
- Shares a major drawback with HOD: a possible significant waste of resources in the Reduce phase, or in the case of having unbalanced executions of Map tasks
  - Because the size of the on-the-fly cluster is statically determined at the beginning by users, normally according to the number of Map tasks
  - On-the-fly Hadoop cluster is not shared across job submissions
System Design - Overview

Client

Serial Job

Reserved Job

MapReduce Job

Monitor

Check status

Hbase Tables

Job Table

User Table

Queue Table

Reservation Table

Job Broker i

poll

Job Broker j

Submit Wrapper Execute Job

Submit Wrapper Execute Job

Wrapper n

Wrapper y

Job x

Pool of MapReduce Worker Nodes
System Design – Client

- **Clients** accept job submission commands from users
- Check user credentials, determine queue/job policy, add job to CloudBATCH system by inserting job information into an HBase table called “Job Table”

<table>
<thead>
<tr>
<th>JobID</th>
<th>QueueID: serial</th>
<th>QueueID: bioinformatics</th>
<th>JobType:serial</th>
<th>JobType:MapReduce</th>
<th>JobType:Reserved</th>
<th>Priority:1</th>
<th>Priority:5</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td>1200000</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>t4</td>
<td></td>
<td>t2</td>
<td>600000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td>t5</td>
<td>1200000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JobID</th>
<th>UserAlice</th>
<th>UserBob</th>
<th>Groupdefault</th>
<th>Groupbio</th>
<th>Command</th>
<th>JobDescription</th>
<th>AbsolutePath1</th>
<th>AbsolutePath2</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>Command1</td>
<td>mpeg encoder</td>
<td>RelativeFileDir1</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td>Command2</td>
<td>bio sequence</td>
<td></td>
<td>RelativeFileDir2</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td>Command3</td>
<td>space weather</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
System Design – Broker and Monitor

- **Brokers** constantly poll the Job Table for a list of jobs with “Status:submitted” status
- For every job on the list
  - Changes the job status to “Status:queued”
  - Submits to the Hadoop MapReduce framework a “Wrapper” program for job execution
  - **Note:** if the Wrapper fails directly after being submitted, jobs could stay in “Status:queued” forever. This is handled by the “Monitor” program
- **Monitors** set a time threshold T, periodically poll the Job Table for jobs that stay in the “Status:queued” status for a time period longer than T, and change their status back to “Status:submitted”
System Design – Wrapper

- **Wrappers** are Map-only MapReduce programs acting as agents to execute user programs at compute nodes where they are scheduled to run by the underlying Hadoop schedulers.

- When a Wrapper starts at some compute node:
  - Grabs from the HBase table the necessary job information and transfers files that need to be staged to the local machine.
  - Updates the job status to “Status: running”.
  - Starts the job execution through *commandline invocation*, MapReduce and legacy jobs alike.

- During job execution:
  - Checks the execution status such as total running time to follow job policies, and terminates a running job if policies are violated.

- After job execution completes:
  - Either successful or failed, the Wrapper will update the job status (“Status: successful” or “Status: failed”) in the Job Table,
  - Performs cleanup of the temporary job execution directory,
  - Terminates itself normally.
System Design - Discussion

- Limitations
  - No support for MPI-type applications
  - Jobs must be commandline based, no interactive execution

- Data consistency under concurrent access
  - Guard against conflicting concurrent updates to Job Table
    - Multiple Brokers updating conflicting job status
  - Solution: use transactional HBase with snapshot isolation

- Performance bottleneck
  - Multiple number of system components (Brokers, etc.) can be run according to the scale of the cluster and job requests
  - The bottleneck is at how fast concurrent clients can insert job information to the Job Table
Future Work

- Monitors are still not fully explored for our prototype and may be extended in the future for detecting jobs that have been marked as “Status:running” but actually failed.

- Further test the system under multi-queue, multi-user scenarios with heavy load and refine the prototype implementation for trial production deployment in solving real-world use cases.

- Exploit the usage of CloudBATCH to make dedicated Hadoop clusters useful for the load balancing of legacy batch job submissions to other coexisting traditional clusters.
Thank you! And Questions?

• Please contact me at:
  • c15zhang@cs.uwaterloo.ca

• Thank you!