CLOUDKON: A CLOUD ENABLED DISTRIBUTED TASK EXECUTION FRAMEWORK

Iman Sadooghi
Dr. Ioan Raicu
Data Intensive Computing (DataSys) Laboratory
Introduction

- **MTC: Many-Task Computing**
  - Bridge the gap between HPC and HTC
  - Many resources over short time periods
  - Loosely coupled apps with HPC orientations
  - Example: MapReduce

- **Data analytics moving towards fine granular tasks**
  - Example: GAMESS(chemistry), TPC-H(industry)

- **Traditional Batch Schedulers**
  - Heavy weight
  - Cannot scale for the new workloads

Image taken from: Sparrow: Scalable scheduling for sub-second parallel jobs. Tech. Rep. UCB/EECS-2013-29, University of California, Berkeley,
Introduction

- **Large Scale Task Execution**
  - Run on distributed resources

- **Workloads**
  - Tasks
    - More in number
    - Shorter in length

- **Requirements for high performance**
  - Concurrency
  - Load Balance
  - System Utilization
Motivation

- Current resources
  - Clusters & Super Computers
  - Alternatives?!
- How about Clouds?
  - Large resources
  - Easier access than the other two
  - Scale up as much as you want
  - Customizable
  - Pay-as-you go model, pay only when you use it
  - Perfect for medium size projects with limited budget
    - Use as long as you have budget
State-of-the-art job schedulers

- Centralized Master/Slaves architecture
  - Scalability issues at petascale and beyond
  - Single point of failure
  - Example: SLURM, CONDOR, Falkon

- Distributed Architectures
  - Hierarchical
    - several dispatchers in a tree-based topology
    - Example: Distributed Falkon
  - Fully distributed
    - each computing node maintains its own job execution
    - Example: Sparrow

- Common issues
  - Poor load balancing
  - Poor system utilization
State-of-the-art job schedulers

- Centralized Master/Slaves architecture
  - Scalability issues at petascale and beyond
  - Single point of failure
  - Example: SLURM, CONDOR, Falkon
State-of-the-art job schedulers

- Centralized Master/Slaves architecture
  - Scalability issues at petascale and beyond
  - Single point of failure
  - Example: SLURM, CONDOR, Falkon

- Distributed Architectures
  - Hierarchical
    - several dispatchers in a tree-based topology
    - Example: Distributed Falkon
  - Fully distributed
    - each computing node maintains its own job execution
    - Example: Sparrow

- Common issues
  - Poor load balancing
  - Poor system utilization
State-of-the-art job schedulers

- **Distributed Architectures**
  - Hierarchical
    - several dispatchers in a tree-based topology
    - Example: Distributed Falkon
  - Fully distributed
    - each computing node maintains its own job execution
    - Example: Sparrow

- **Common issues**
  - Poor load balancing
  - Poor system utilization
Agenda

Background

Proposed Work

- CloudKon Architecture
- Task Consistency
- Dynamic Provisioning
- Communication Cost
- Implementation details

Performance Evaluation

- Throughput
- Latency
- Consistency effect on throughput and latency
- Efficiency
- Consistency effect on efficiency

Conclusion and Future work
Amazon Simple Queue Service (SQS)

- Distributed message delivery queue
  - Highly scalable
  - Messages sent and read simultaneously
    - Messages sent to multiple servers
- Reliable
  - Guarantees message delivery
    - At least once delivery
    - Multiple copies may be available and accessed
- Secure
  - Through authentication
Amazon Dynamo DB

- No-SQL Key Value Store
- Fully distributed
- Faster and more scalable than traditional DBs
- Simple query support
- Atomic operations support
  - Atomic read
  - Atomic write
Agenda

- Intro and Motivation (5min)
- Background (2min)
- Proposed Work (6min)
  - CloudKon Architecture
  - Task Consistency
  - Dynamic Provisioning 15s
  - Monitoring 15s
  - Communication Cost 15s
  - Implementation details
- Performance Evaluation (5min)
  - Throughput
  - Consistency effect on throughput and latency
  - Efficiency
  - Consistency effect on efficiency
- Conclusion and Future work (2min)
Proposed Work

• Use SQS as a task delivery component
• Decouple Clients and Workers
• Pushing vs. Pulling approach
  • Pushing
    • Local/global manager node needs to predict/decide
      • Randomness
      • Get system information periodically from workers
      • Needs to know about the address of worker nodes.
  • Pulling
    • No need to know about workers
    • Workers decide for themselves
• Load balancing
• System Utilization
CloudKon Architecture

DP: Dynamic Provisioning
MS: Monitoring System
MT: Monitoring Thread
WT: Worker Thread
CT: Client Thread

DynamoDB

Client

Global Request Queue

Client Response Queues

Worker
Task consistency

• SQS only guarantees at least once delivery
• some workloads require exactly once execution of tasks!
• Use DynamoDB to verify
• Use conditional write
  ➢ Write if the task does not exist
  ➢ Throw exception if exists
• Atomic operation
• Using a single operation, the checking is done
  • Minimize the communication overhead
Dynamic Provisioning

- Dynamically scale up and down the system
- Scale up
  - Use Provisioner component
  - Check request queue length (periodically)
  - Launch new worker if it’s getting larger

- Scale down
  - If:
    - The worker goes idle (because of having no job to run!)
    - The rent time is closer than threshold to the rent unit value of time
  - Then:
    - Terminate the worker instance
  - Benefits:
    - No component needs to keep track of workers
Monitoring

- Monitor workers for:
  - System utilization
  - Debug

- Monitor Thread
  - Each worker thread has a monitor thread
  - Reports system utilization periodically
  - Able to report other details of each worker

- Monitoring System
  - Reads the aggregate utilization results from store
Communication Cost

• Communication overhead is high on Cloud
  • Need to minimize the communication
• Message batching
  • Bundle tasks together to send
• Number of communications
  • Minimum possible number
Implementation Details

- Written in Java
- Dependency
  - AWS Java SDK library
  - Apache Commons library
  - Google protocol buffer library
- Serialization
  - Used Google Protocol Buffer
    - More efficient protocol than JSON
- Simple and short code base
  - Only 1052 lines of code
  - Delivers 2X performance with less than 5% code base length

<table>
<thead>
<tr>
<th></th>
<th>CloudKon</th>
<th>Sparrow</th>
<th>Falkon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>1052</td>
<td>24500</td>
<td>33000</td>
</tr>
</tbody>
</table>
Agenda

■ Background
■ Proposed Work
  ▪ CloudKon Architecture
  ▪ Task Consistency
  ▪ Dynamic Provisioning
  ▪ Communication Cost
  ▪ Implementation details

■ Performance Evaluation
  ▪ Throughput
  ▪ Consistency effect on throughput
  ▪ Efficiency
  ▪ Consistency effect on efficiency

■ Conclusion and Future work
Throughput

- 1 to 64 instances
- 16000 to 1024000 tasks
- 5735 msgs/sec on the largest scale (64)
Latency

- 24.6 ms latency on 64 scale
  - Compared to 49.9 ms and 125.5 ms
- Duplicate task controller enabled/disabled
- 30% overhead on average
- Overhead decreasing on larger scales
Consistency effect on latency

- 37% overhead on average
**Efficiency**

- 64 instances scale
- High efficiency on 1 sec tasks (91.26%)
- Moderate efficiency on tasks with 100s of ms length.
Consistency effect on efficiency

- Duplicate task controller enabled/disabled
- Overhead decreasing on larger scales
Agenda

- Background
- Proposed Work
  - CloudKon Architecture
  - Task Consistency
  - Dynamic Provisioning
  - Communication Cost
  - Implementation details
- Performance Evaluation
  - Throughput
  - Latency
  - Consistency effect on throughput and latency
  - Efficiency
  - Consistency effect on efficiency
- Conclusion and Future work
Conclusion

- Design and implement simple yet effective distributed task execution framework
  - Using cloud services like SQS, DynamoDB
- Run on Public Cloud environment as an alternate resource
  - Optimum usage of cloud resources
- Outperforming other state of the art systems
  - Sparrow 2013
  - Falkon 2007
    - High throughput and efficiency
Future work

• On Cloud Environment
  • Extend the evaluation scale to 1024 instances
  • Run real applications on CloudKon
    • Industrial benchmarks: TPC-H
    • Data Analytics: MapReduce applications (Hadoop workloads)
  • Implement a SQS like service
    • Using ZHT distributed hash table as a building block
    • Make CloudKon infrastructure independent
    • Test CloudKon on private clouds (e.g. OpenStack)

• On HPC environment
  • Create a tightly coupled system using our own Distributed Queue implementation
    • Deliver lower latency
  • Evaluate the performance on HPC Clusters and super computers
    • Run real applications
Thank you

- Questions?!