MATE-EC2: A Middleware for Processing Data with Amazon Web Services

Tekin Bicer  David Chiu* and Gagan Agrawal

Department of Computer Science and Engineering
Ohio State University

* School of Engineering and Computer Science (Presenter)
Washington State University, Vancouver

The 4th Workshop on Many-Task Computing on Grids and Supercomputers: MTAGS’11
Today’s applications are increasingly data and compute intensive

Many-Task Computing paradigms becoming pervasive: WF, MR

E.g., Map-Reducible applications are solving common problems
  – Data mining
  – Graph processing
  – etc.
• Infrastructure-as-a-Service
  – Anyone, anywhere can allocate “unlimited” virtualized compute/storage resources

• Amazon Web Services:
  – Most popular IaaS provider
  – Elastic Compute Cloud (EC2)
  – Simple Storage Service (S3)
On-Demand Instances (Virtual Machines)
- Types: Extra Large, Large, Small, etc.

For example, Extra Large Instance:
- Cost: $0.68 per allocated-hour
- 15GB memory; 1.7TB disk (ephemeral)
- 8 Compute Units
- High I/O performance
Amazon Web Services: S3

- Accessible anywhere. High reliability and availability
- Objects are arbitrary data blobs
- Objects stored as \((\text{key}, \text{value})\) in \textit{Buckets}
  - 5TB of data per object
  - Unlimited objects per bucket
Amazon Web Services: S3 (Cont.)

- Simple FTP-like interface using web service protocol
  - Put, Get (Partial Get), and Delete
  - SOAP and REST

- High throughput (~40MB/sec)
  - Scales well to multiple clients

- Low costs
Amazon Web Services: S3 (Cont.)

- 449 billion objects in S3 as of July 2011
  - Doubling each year
• Virtualization is characteristic of any cloud environment: Clouds are **black boxes**

• Storage and elastic compute services exhibit performance variabilities we should leverage
Goals

• As users are increasingly moving to cloud-based solutions for computing....

• We have a need for services and tools that can...
  – Get the most out of cloud resources for data-intensive processing
  – Provide a simple programming interface
Outline

• Background
• System Overview
• Experiments
• Conclusion
MATE-EC2 System Design

• Cloud middleware that is able to
  – Use a set of possibly heterogeneous EC2 instances to *scalably* and *efficiently* process data stored in S3

• MATE is a *generalized reduction* PDC structure like Map-Reduce
### MATE and Map-Reduce

#### Map-Reduce

```java
// outer sequential loop
while () {
    // reduction loop
    for each (element e) {
        (i, val) := process(e);
    }
    sort (i, val) pairs over i
    reduce to compute each rObj(i)
}
```

#### MATE

```java
// outer sequential loop
while () {
    // reduction loop
    for each (element e) {
        (i, val) := process(e);
        rObj(i) := reduce(rObj(i), val);
    }
    global reduction to combine rObjs
}
```
MATE-EC2 Design
Objects: Physical representation of the data in S3

Chunks: Logical data partitions within objects (exploits memory utilization)

Units: Fixed data units within a chunk for retrieval (exploits concurrency)

Metadata: chunk offset, chunk size, unit size
Threaded chunk retrieval: Chunk retrieved concurrently with a number of threads
Dynamic Load Balancing

S3 Data Retrieval Layer

Buffer

Th₀

Th₁

Th₂

Th₃

input

Computing Layer

EC2 Slave Instance

S3 Data Object

 CHUNK₀

...

 CHUNKₙ

S3 Data Store

Job Scheduler

EC2 Master Instance

Metadata

Job Pool
Dynamic Load Balancing

Simple greedy heuristic:
Select a unit belonging to the least connected chunk
MATE-EC2 Processing Flow

(1) Compute node requests a job from Master
(2) Chunk retrieved in units
MATE-EC2 Processing Flow

(3) Pass to Compute Layer, and process
MATE-EC2 Processing Flow

(4) Request another job from Master
MATE-EC2 Processing Flow

Slave Instance

EC2 Slave Instance

Slave Instance

Slave Instance

S3 Data Retrieval Layer

Computing Layer

EC2 Master Instance

Job Scheduler

Metadata

S3 Data Object

S3 Data Store

Th0

Th1

Th2

Th3

Chunk0

Chunkn
Experiments

• Goals
  – Finding the most suitable setting for AWS
  – Performance of MATE-EC2 on heterogeneous and homogeneous compute environments
  – Performance comparison of MATE-EC2 and Map-Reduce
Experiments (Cont.)

• Setup:
  – 4 Large EC2 slave instances
  – 1 Large instance for master instance
  – For each application, the dataset is split into 16 data objects on S3

• Large Instance:
  – 4 compute units (each comparable to 1.0-1.2GHz)
  – 7.5GB (memory)
  – 850GB (disk, ephemeral)
  – High I/O
## Experiments (Cont.)

<table>
<thead>
<tr>
<th>App</th>
<th>I/O</th>
<th>Comp</th>
<th>RObj Size</th>
<th>Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KMeans Clustering</strong></td>
<td>Low/Med</td>
<td>Med/High</td>
<td>Small</td>
<td>8.2GB 10.7 billion points</td>
</tr>
<tr>
<td><strong>PCA</strong></td>
<td>Low</td>
<td>High</td>
<td>Large</td>
<td>8.2GB</td>
</tr>
<tr>
<td><strong>PageRank</strong></td>
<td>High</td>
<td>Low/Med</td>
<td>Very Large</td>
<td>1GB 9.6M nodes, 131M edges</td>
</tr>
</tbody>
</table>
Effect of Chunk Sizes (KMeans)

- Performance increase:
  - 128KB vs. >8M
  - 2.07x to 2.49x speedup
Data Retrieval (KMeans)

**128M Chunk Size**

- One Thread vs. others: 1.37x - 1.90x

**16 Data Retrieval Threads**

- 8M vs. others speedup: 1.13x - 1.30x
Job Assignment (KMeans)

- Speedup:
  - 1.01x for 8M
  - 1.1x to 1.14x for others
MATE-EC2 vs. Elastic MapReduce

Chunk Size: 128MB
Data retrieval Threads: 16

**KMeans**

- Speedups vs. EMR-combine: 3.54x to 4.58x

**PageRank**

- Speedups vs. EMR-combine: 4.08x to 7.54x
MATE-EC2 on Heterogeneous Instances

(a) KMeans – 128MB Chunk Size, 16 Data Retrieval Threads
(b) PCA – 128MB Chunk Size, 16 Data Retrieval Threads

• Overheads
  – KMeans: 1%
  – PCA: 1.1%, 7.4%, 11.7%
In Conclusion...

- AWS environment is explored for data-intensive computing
  - 64M and 128M data chunks w/ 16 data retrieval threads seems to be optimal for our middleware

- Our data retrieval and processing optimizations significantly improve the performance of our middleware

- MATE-EC2 outperforms MR both in scalability and performance
Thank You

• Questions & Discussion

• Acknowledgments:
  – We want to thank the MTAGS’11 organizers and the reviewers for their constructive comments

  – This work was sponsored by: