Detecting Bottlenecks in Parallel DAG-based Data Flow Programs

Björn Lohrmann

Dominic Battré
Matthias Hovestadt
Alexander Stanik
Daniel Warneke

Email: {firstname}.{lastname}@tu-berlin.de

Complex and Distributed IT-Systems
Technische Universität Berlin
IaaS clouds offer virtual machines on-demand

Why use clouds for data processing?
- Fast and unlimited** scale-out
- Pricing Model
  - Pay-as-you-go
  - 10 nodes for 1 day = 1 node for 10 days
- No long-term obligations

**almost
Frameworks are required for effective use of clouds

Job Modelling
Parallelization
Job Scheduling
VM Management
Job Monitoring
Job Deployment

Eucalyptus
Hadoop
Nephele
etc.
Prerequisites

- Jobs modelled as directed acyclic graphs
  - Vertices are tasks
  - Edges are communication channels
- Each task has 1..n parallel task instances
- Unidirectional and blocking communication
Overview

Key question of this talk:
- Given a DAG-shaped job, how many task instances should I assign to each task?

Our approach
- Begin with 1 instance for each task
- Iteratively detect bottlenecks and add instances where necessary
Bottlenecks

Negative effects of bottlenecks:

- Input starvation
- Output blockage

- Low throughput of workflow
- Low resource utilization
- Time and money wasted
Bottlenecks

Types:

- **CPU**
  - Enough input available
  - Throughput limited by CPU
  - Lack of input for subsequent tasks

- **I/O**
  - Transport infrastructure is overloaded (NICs, switches, etc)
  - Forces tasks to wait
Bottleneck Detection

● Monitor job at runtime:
  ● Continuously measure CPU load and I/O wait on task instances
  ● Aggregate to task statistics

● Continuously analyze task statistics:
  ■ Traverse task nodes in reverse topological order and check for CPU bottlenecks
  ■ If none found traverse edges in reverse topological order and check for I/O bottlenecks
  ■ If bottleneck found: Report it!
Implementation

- Based on Nephele framework
  - Java framework
  - 1 master, n workers
  - Task instance = Java thread

- Analysis of thread state statistics:
  - Threshold for CPU bottleneck:
    - USR + SYS + BLK >= 90% time
  - Threshold for I/O bottleneck
    - WAIT caused by sending on channel >= 90% time
Evaluation

Demo Job

- PDF Writer
- Index Writer
- PDF Creator
- Inverted Index
- OCR
- File Reader

Setup:
- Private compute cloud
- Hosts with two Intel Xeon 2.66Ghz, 32 GB RAM and 1GB Ethernet
- KVM guests with one virtual CPU and 2GB RAM
- Eucalyptus framework for VM allocation/deallocation
Phase 1: Fine tuning
Evaluation (1)

Phase 2: Scale-out
Conclusion

- Bottleneck detection is useful to scale out jobs in the cloud, while maintaining high resource utilization
- We presented a simple approach to gather and analyze relevant statistics
- Right now, manual adaptation and job re-runs are necessary to eliminate bottlenecks

Future work:
- Dynamically and automatically adjust parallelization at runtime