CloudKon: DTS – Reloaded with efficient Monitoring ,Bundled Response and Dynamic Provisioning with improved concurrency.

–Sandeep Palur and Ajay Anthony
(A20302187) (A20306352)

Contents

Introduction to Cloudkon

Cloudkon Architecture

Cloudkon Reloaded Architecture

Cloudkon Reloaded Improvements

Benchmarking results

Conclusion

Contributions

Demo

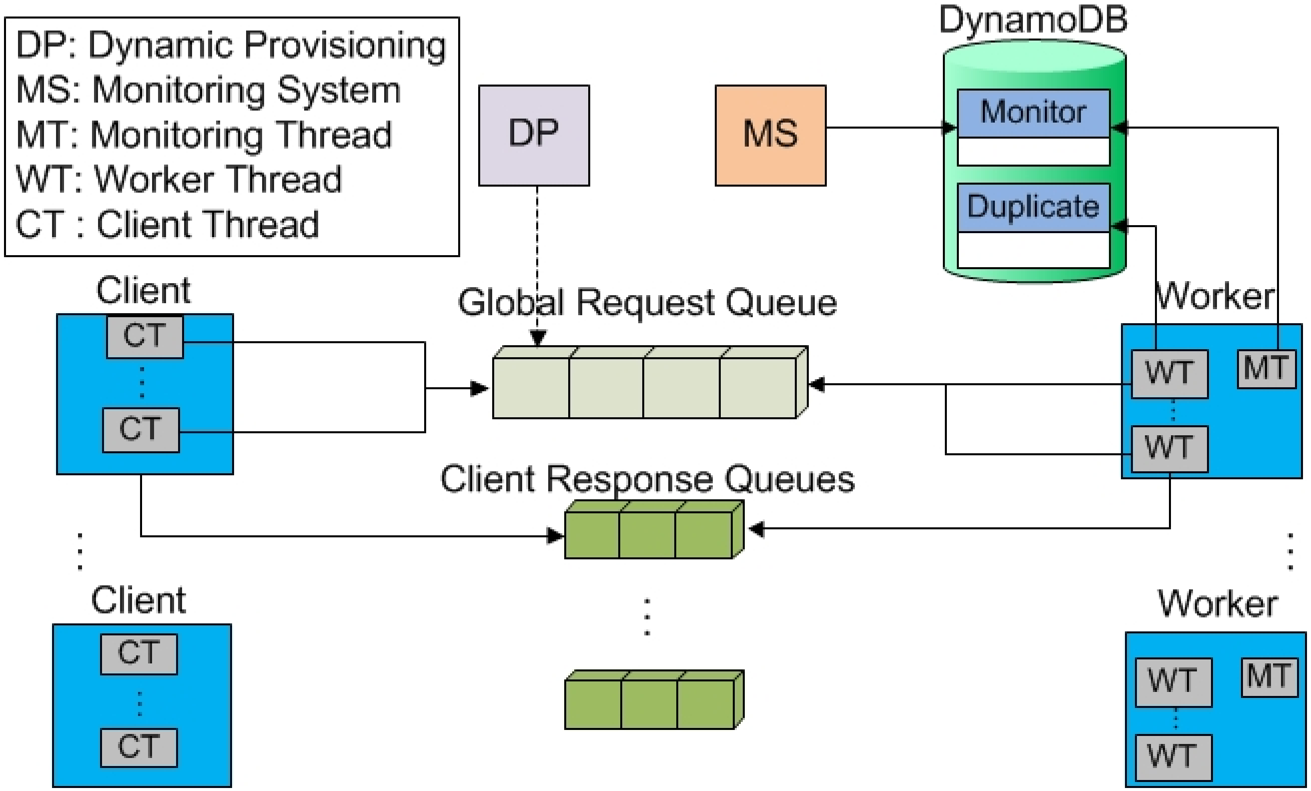
Introduction

- CloudKon is a compact, light-weight, scalable, and distributed task execution framework .

- Built on following Amazon components:
- EC2
- SQS
- DynamoDB

- Major Components in CloudKon:
- Client
- Server
- Global Request Queue (SQS)
- Client Response Queue (SQS)

Cloudkon Architecture

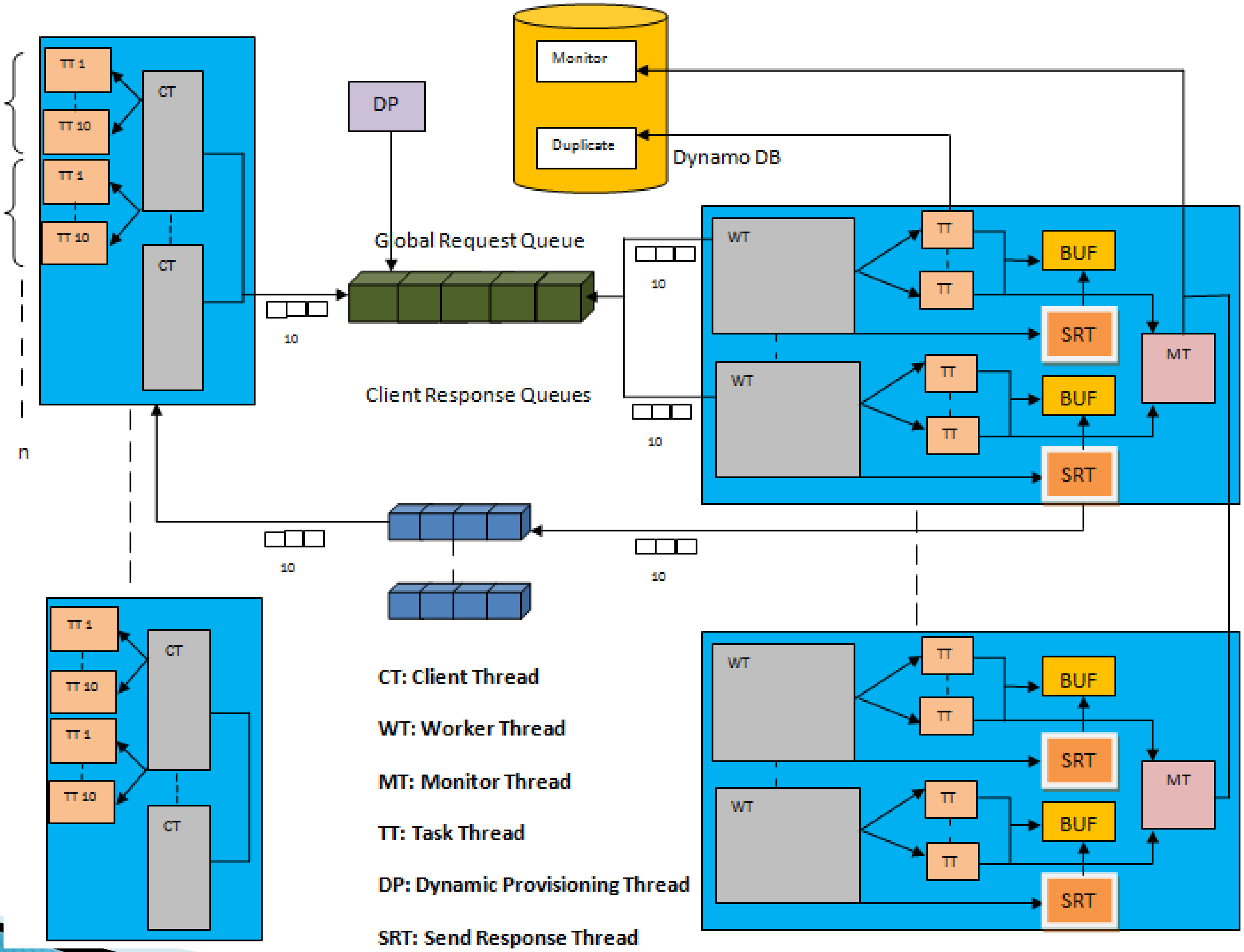


DP: Dynamic Provisioning
MS: Monitoring System
MT: Monitoring Thread
WT: Worker Thread
CT : Client Thread

CloudKon-Reloaded Improvements

- 1. Improved concurrency
- 2. Bundled Response
- 3. Efficient Monitoring

CloudKon-Reloaded Architecture



CT: Client Thread

WT: Worker Thread

MT: Monitor Thread

TT: Task Thread

DP: Dynamic Provisioning Thread

SRT: Send Response Thread

CloudKon-Reloaded Components

- **Server**
- **Worker Thread (WT)**

1. Pulls task bundles from global request queue.
2. Creates task thread in optimal concurrency mode.

- **Task Thread (TT)**

1. Deletes the task from the global request queue.
2. Checks for duplication with DynamoDB.
3. Executes task and puts back response to client specific array in Buffer.

CloudKon-Reloaded Components

- **Buffer (BUF):**

  1. Concurrent hash map
     - Key :Client Response Queue link.
     - Value: ArrayList of task responses.

- **Send Response Thread (SRT):**

  1. Pulls message bundles from buffer.
  2. Sends bundled response to clients.

- **Monitor Thread (MT):**

  1. Attaches object with task thread.
  2. Tracks utilization using object's reference.

CloudKon-Reloaded Components

- **Client**
- **Worker Thread (WT):**

  1. Creates client response queue.
  2. Submits task to global request queue.
  3. Pulls messages from it’s response queue.
  4. Creates task threads using maximum concurrency mode.

- **Task Thread (WT):**

  1. Deletes message from response queue.
  2. Adds message in the concurrent ArrayList.

Improvements

1. Improved concurrency

- All tasks are processed concurrently.
- Reduces Latency.
- Increases throughput.

2. Bundled Response:

- Reduces network overhead.
- Utilizes network bandwidth more effectively i.e. reducing the probablity of network latency.

3. Efficient Monitoring:

- Reduces network overhead .
- Reduces contention by 1/n, where n = no. of workers

Benchmarking

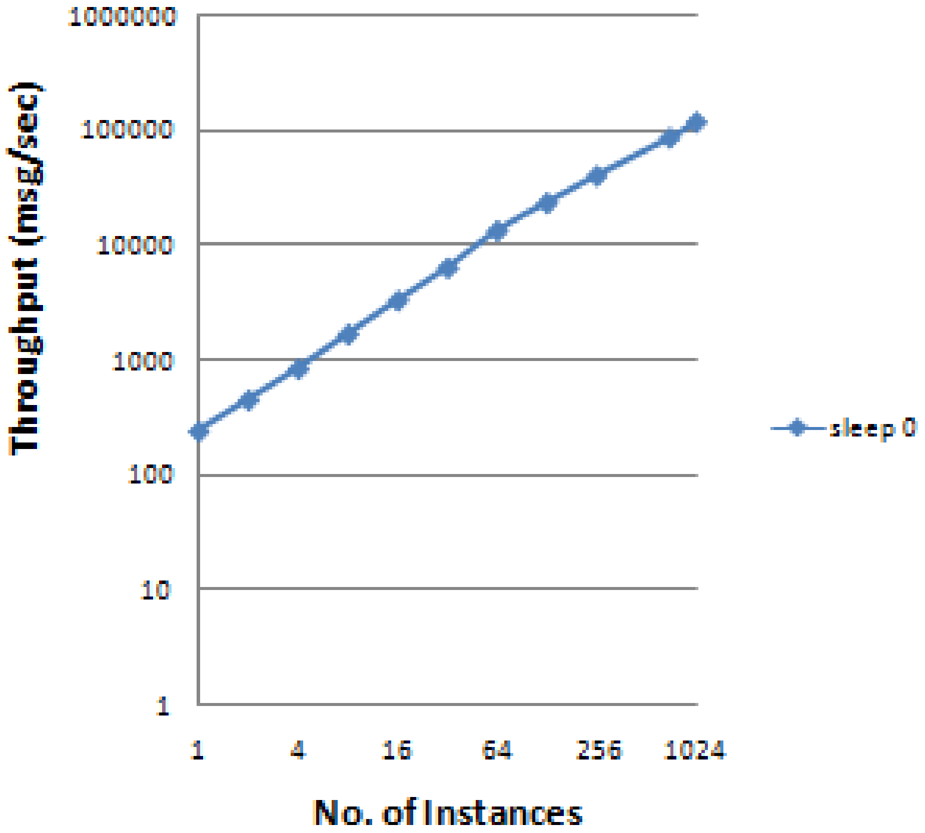
- **Test-bed:**
  - Ran on Amazon EC2 instances experiments on us.east.1 datacenter of Amazon.
  - Instance type - m1.large
  - All instances have Linux OS with JRE 1.7 installed.
  - Each instance runs both client and server.
  - 2 client threads and 4 worker threads run on each instance.
  - Each instance submits 16000 tasks. (8000/thread)
  - Tasks: sleep 0 , 16, 128

Benchmarking

- *Scripts and programs developed specifically for benchmarking:*
- 1. Shell Scripts (Bash): Throughput, Latency, File transfer from EC2 instances.
- 2. Parallel-SSH: For parallel execution on EC2.
- 3. EC2 CLI (Command Line Interface): For instance startup, terminate, Getting IP address, etc.
- 4. AWS CLI (Command Line Interface): Mainly for Dynamic Provisioning for SQS operations and EC2 dynamic instance startup.

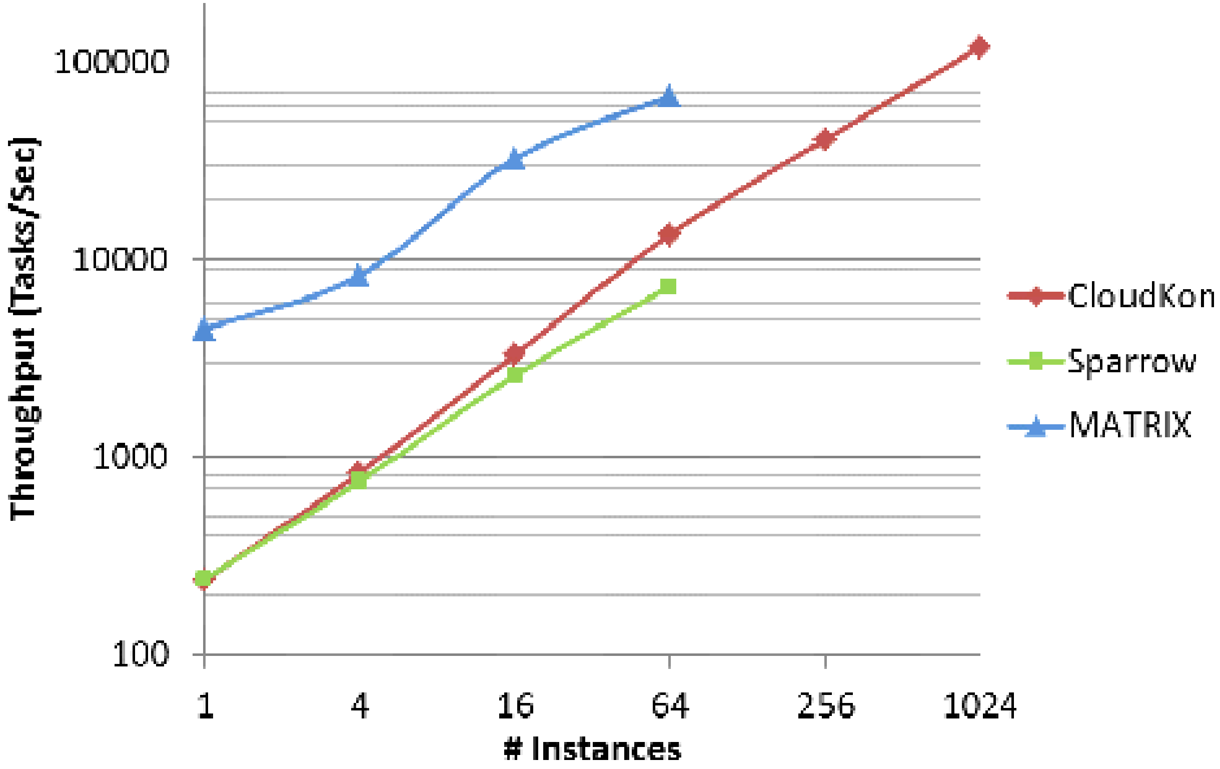
Benchmarking

- Throughput:
  - sleep 0 tasks



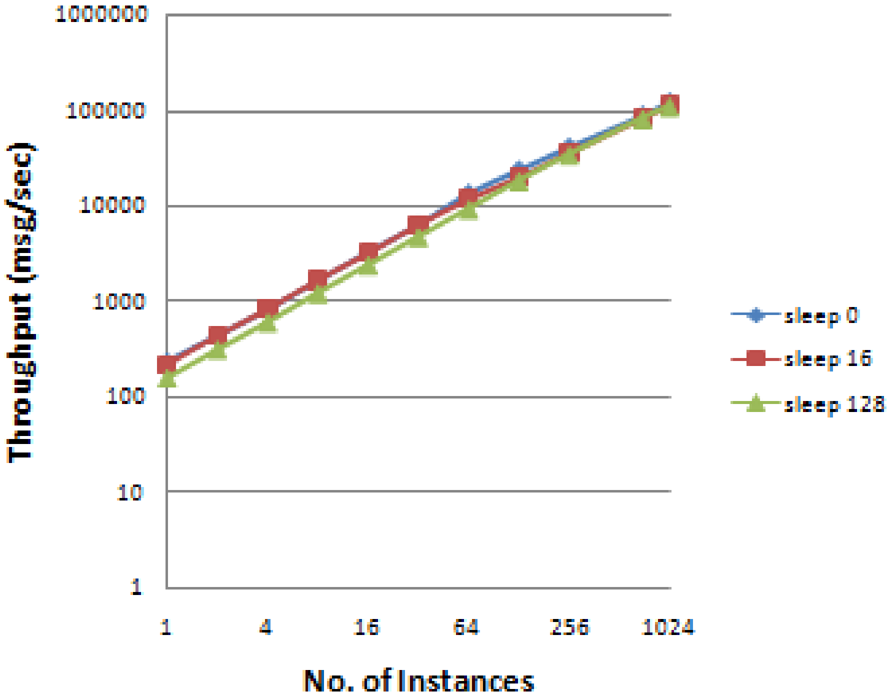
Benchmarking

- Throughput Comparison:



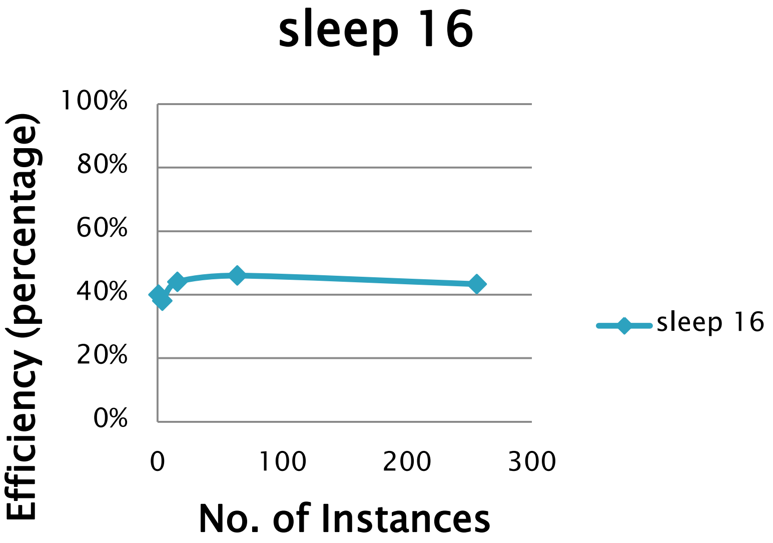
Benchmarking

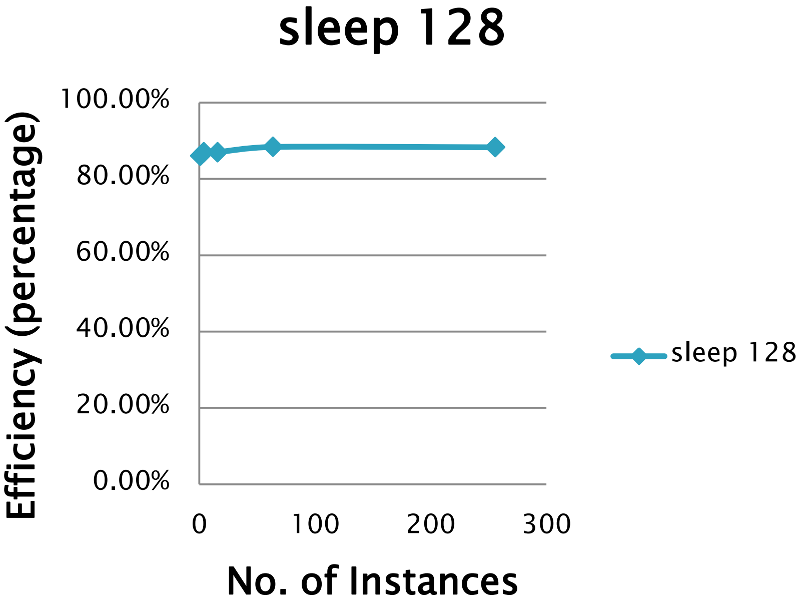
- Comparison of Sleeps for Throughput:
- sleep 0 tasks



Benchmarking

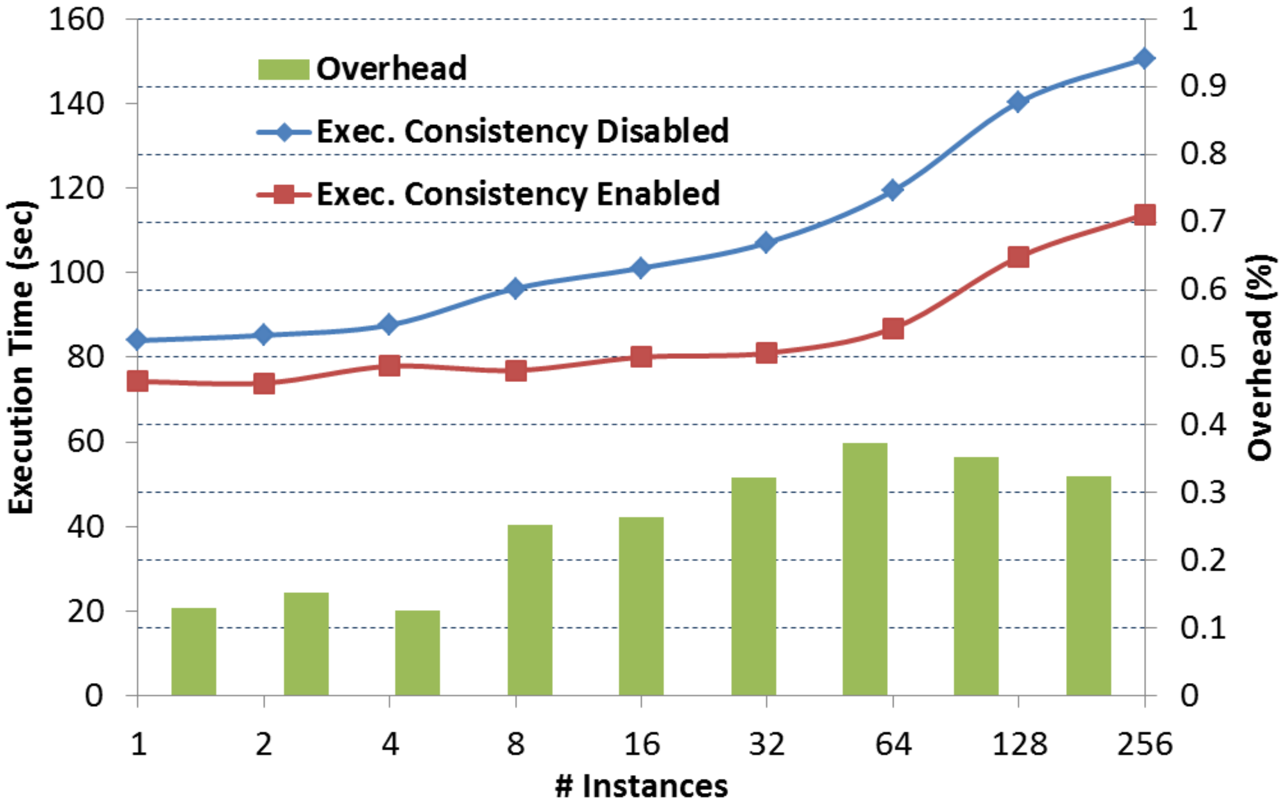
- Efficiency:
  - Homogenous workloads.





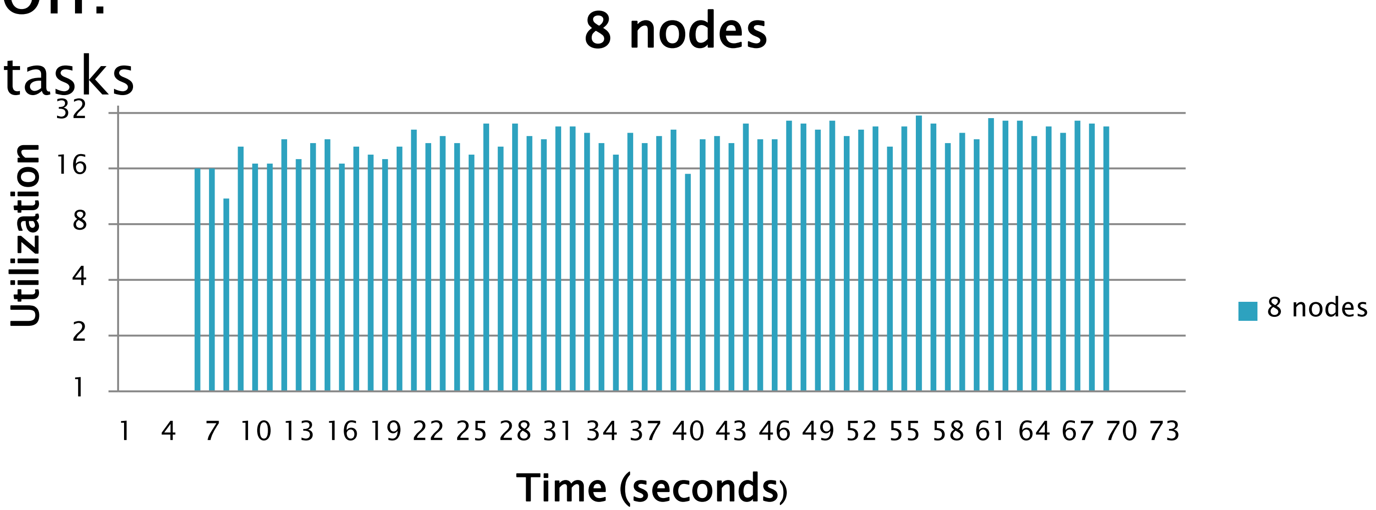
Benchmarking

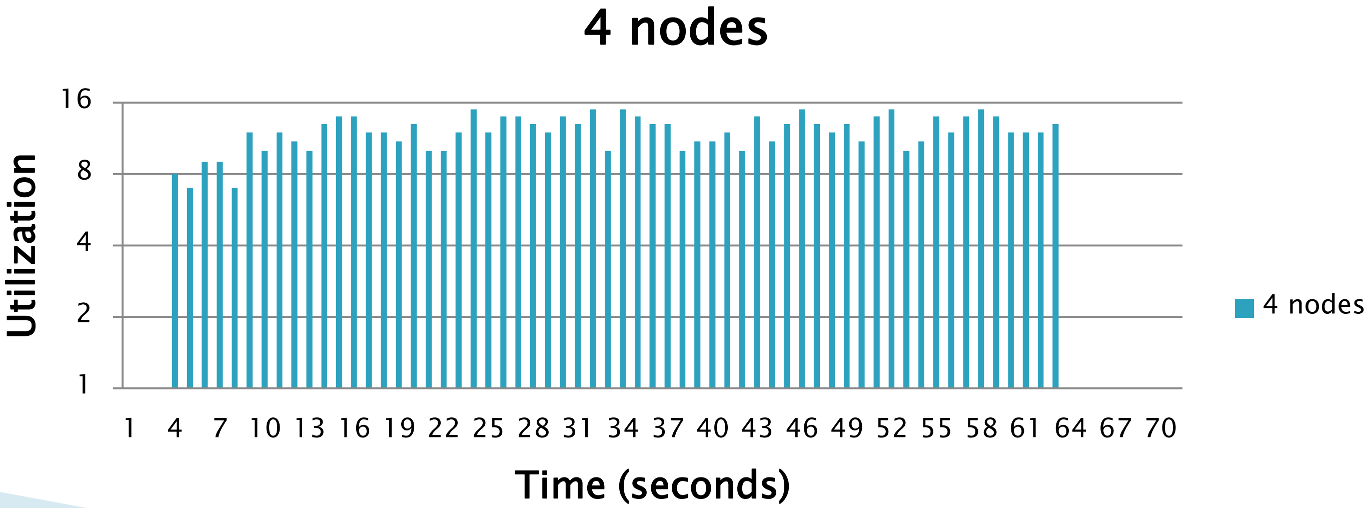
- Consistency:
- sleep 16 tasks



Benchmarking

- Utilization:
  - sleep 100 tasks





Conclusion

- The evaluation of the CloudKon proves that it is highly scalable and achieves a stable performance over different scales.
- CloudKon achieves up to 87% efficiency.
- CloudKon was able to outperform other systems like Sparrow and MATRIX on scales of 128 instances or more in terms of throughput.

Contributions

- Throughput and efficiency experiments for sleep (0,1,16,128) on the following scales (1,2,4,8,16,32,64,128,256,512,1024).

- Our code was used for throughput and efficiency benchmarking experiments in CloudKon paper submitted for CCGRID 2014.

DEMO

THANK YOU

Questions??