The Bakery++ Algorithm

The Bakery algorithm is the first true solution of mutual exclusion, but it suffers from register overflows.

Bakery++ is a slightly modified version of Bakery that avoids overflows without introducing new variables or redefining the operations or functions of Bakery.

Bakery++ is quite simple.

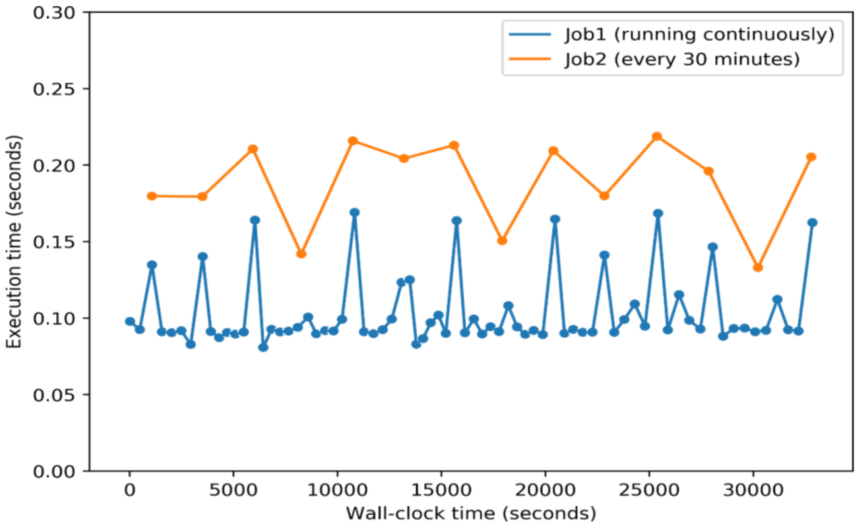
Bakery++ is specified formally in the PlusCal language and verified correct using the TLC model checker.

Communication-aware Job Scheduling using SLURM

Priya Mishra, Tushar Agrawal, Preeti Malakar
Indian Institute of Technology Kanpur

MOTIVATION

Performance of communication-intensive jobs affected by network contention, node-spread and job interference



OBJECTIVE

Developing node-allocation algorithms that consider the job's behaviour during resource allocation to improve the performance of communication-intensive jobs

METHODS

- **Greedy Allocation:** Nodes allocated on switches with lower communication ratio (lower contention and higher free nodes)
- **Balanced Allocation:** Nodes allocated in powers-of-two to minimize inter-switch communication
- **Adaptive Allocation:** Selects the more optimal node-allocation algorithm (greedy or balanced) based on their cost of communication

RESULTS

- Proposed algorithms reduce the execution times by **9%** on average and the wait times by **31%** across three job logs
- Balanced and adaptive always perform better than default and greedy
- Proposed algorithms always perform better than the default for the same cluster state *(individual runs)*

Characterizing the Cost-Accuracy Performance of Cloud Applications

Sunimal Rathnayake, *Lavanya Ramapantulu, Yong Meng Teo
National University of Singapore, *Nanyang Technological University

Motivation

- scalable
- resource pool
- pay for use charging

cloud resources

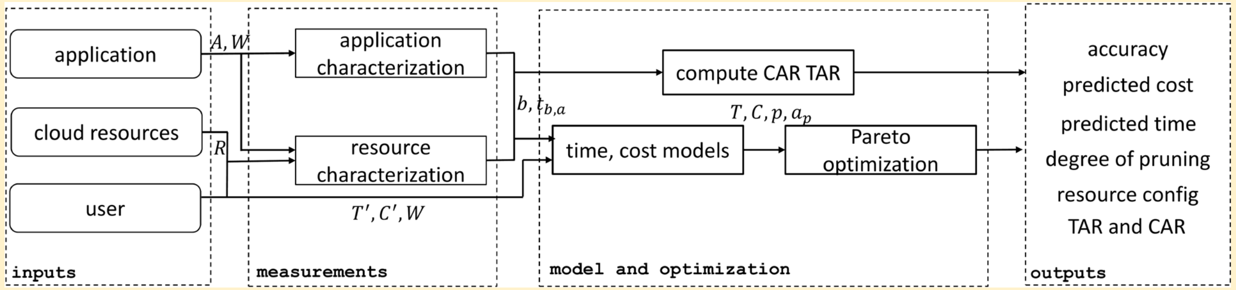
- results of different accuracy
- resource demand varies with accuracy
- e.g. machine learning

some cloud applications

Opportunity for Trading-off Accuracy for Time and Cost

Approach

two stage approach
- measurements for characterization
- model and optimization for determining cost, time and configuration



Contribution

- Measurement-driven model and analysis
- Cost-accuracy "sweet spots"
- Cost-accuracy and time-accuracy Pareto optimal configurations
- Metrics for cost-accuracy and time-accuracy performance

SRMPDS 2020

Scheduling Task-parallel Applications in Dynamically Asymmetric Environments

Jing Chen, **Pirah Noor Soomro**, Mustafa Abduljabbar, Madhavan Manivannan, Miquel Pericàs

Motivations

- Applications sharing resources suffer from interference.
- Runtime scheduling techniques coupled with application knowledge can be used to mitigate interference.
- An online performance model is used to predict task performance.
- We leverage task moldability and knowledge of task criticality to adapt to interference.
- Our scheduler targets to minimize resource usage, execution time and overcommitting of resources.

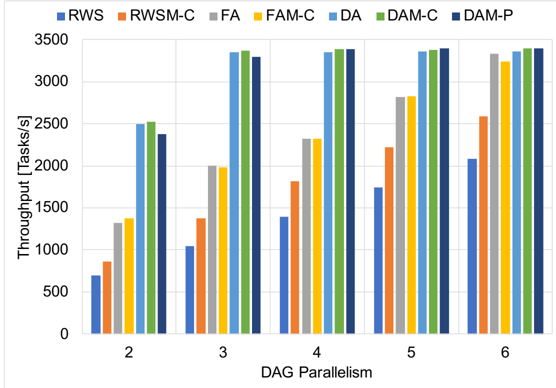
Method

Performance Trace Table (PTT)

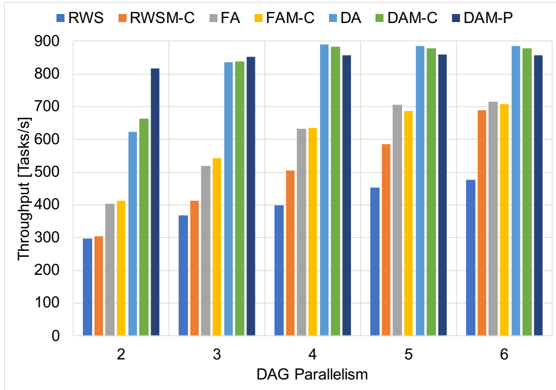
	RW=1	RW=2	RW=4
C0	EP(0,1)	EP(0,2)	EP(0,4)
C1	EP(1,1)		
C2	EP(2,1)	EP(2,2)	
C3	EP(3,1)		

- **Goal:** Performance prediction for future tasks given a set of resources;
- Entries: elastic execution place (leader core, resource width);
- One PTT for each task type;
- Dynamic update of execution time records during execution;
- Awareness of interference activities;
- Only require few information;
- PTT is independent of platforms;
- Low overhead.

Results



Interference: co-running application



Interference: DVFS

Network and Load-Aware Resource Manager for MPI Programs

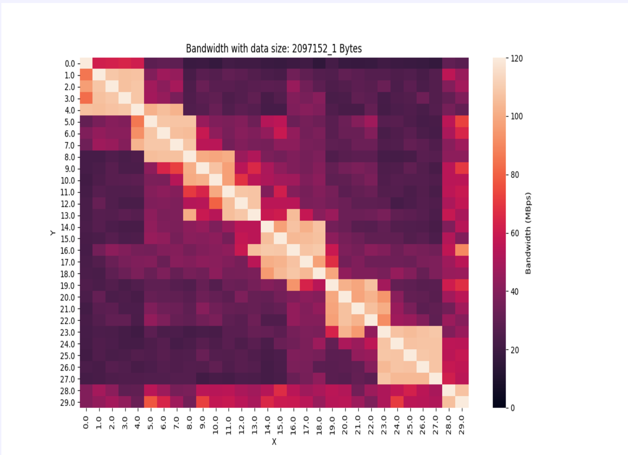
Ashish Kumar, Naman Jain, Preeti Malakar

Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur

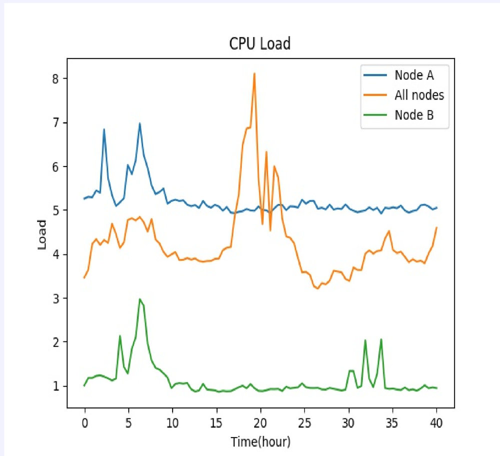
Problem

Node allocation in a shared cluster for parallel jobs to maximize performance considering both compute and network load on the cluster.

Challenges



(a) N/W bandwidth



(b) CPU load

Figure: Variation across nodes

- Non exclusive access to nodes in shared cluster
- Variation in load/utilization across time/nodes
- Topology does not capture the current state of network
- Contention and congestion in the network due to existing jobs
- Varying computation and communication requirements of different programs

Core Components

Resource Monitor

- Distributed monitoring system for cluster
- Uses light-weight daemons for periodically updating livehosts, node statistics and network status

Allocator

- Allocates nodes based on user request
- Considers node attributes and network dynamics
- Uses data collected by resource monitor

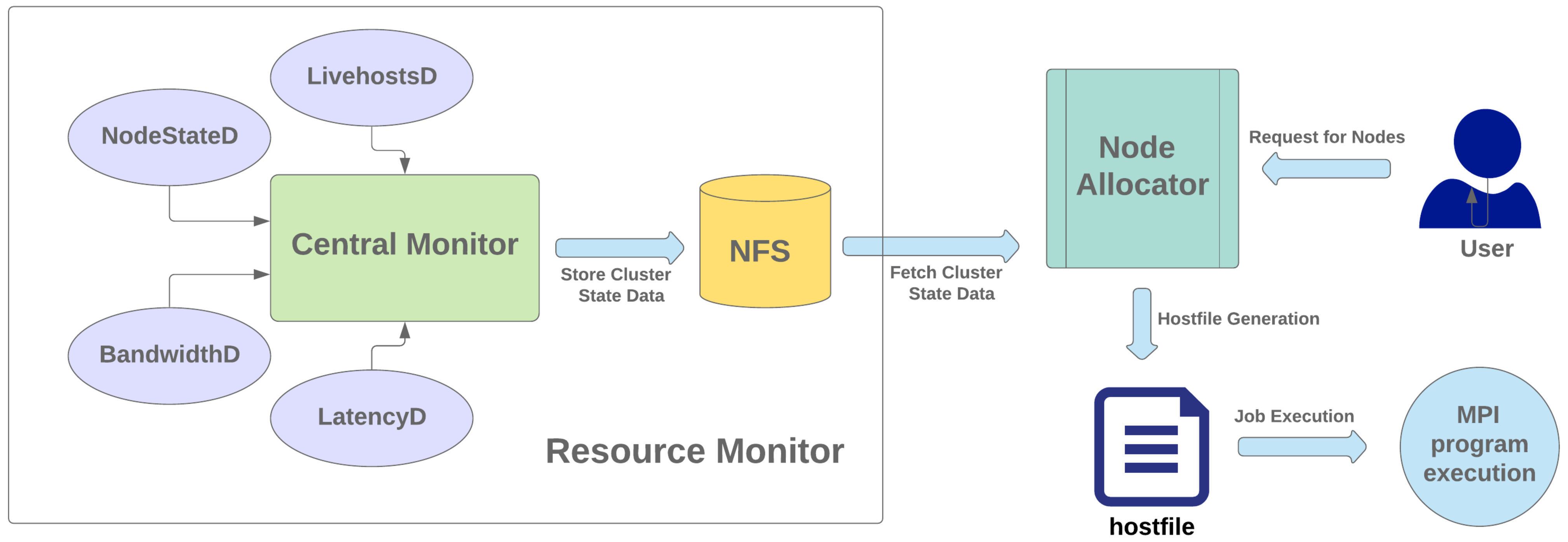


Figure: Allocator workflow

Problem Formulation

Model: Represent cluster as graph with vertices as compute nodes and edges as network links

Objective: Find a sub-graph satisfying user demands such that the overall load of sub-graph is minimized

Compute Load

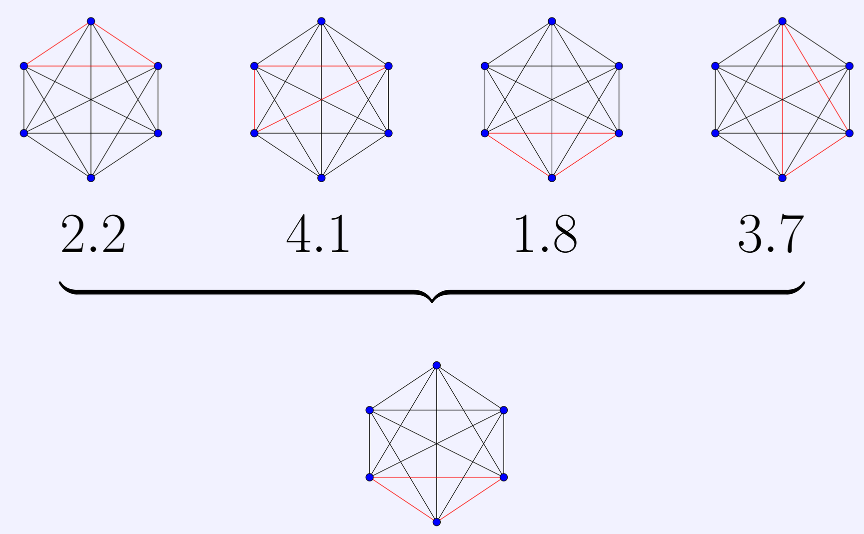
- Measure of overall load on the node
- Static (core count, clock speed) & dynamic (CPU load, available memory) attributes
- $CL_v = \Sigma_{a \in \text{attributes}} w_a * val_{va}$

Network Load

- Measure of load on the P2P network link
- Considers bandwidth and latency
- Topology automatically gets captured
- $NL_{(u,v)} = w_{lt}\text{LT}_{(u,v)} + w_{bw}\text{BW}_{(u,v)}$

Algorithm

- Find candidate sub-graphs
- Calculate total load for each sub-graph
  - Compute Load, $C_{G_v} = \Sigma_{u \in \mathcal{V}_v} CL_u$
  - Network Load, $N_{G_v} = \Sigma_{(x,y) \in \mathcal{E}_v} NL_{(x,y)}$
  - Total Load $= \alpha \times C_{G_v} + \beta \times N_{G_v}$
- Pick the best one according to total load

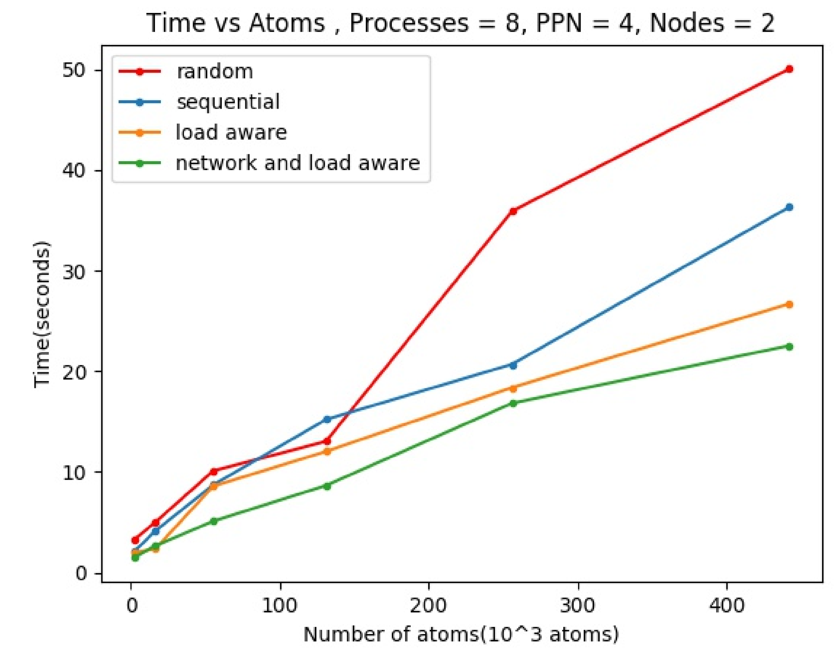


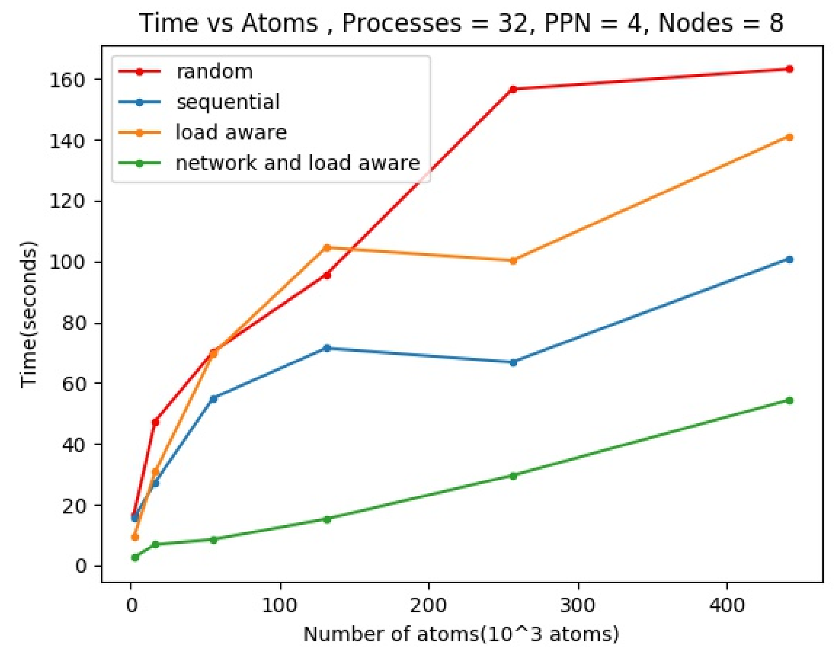
Pictorial representation of allocation algorithm

Candidate Selection Algorithm

- Start with a particular node v
- Calculate addition load for all nodes w.r.t. start node
  $A_v(u) = \alpha \times CL(u) + \beta \times NL(v, u)$
- Keep adding nodes in increasing order of addition load to sub-graph until request is satisfied

Results





Algorithm	Avg. gain	Max. gain
Random	49.9%	87.8%
Sequential	43.1%	84.5%
Load Aware	32.4%	87.7%

Table: Performance gain using our allocation method

Observations

- Our algorithm performs better than random, sequential, and load-aware on an average.
- Load-aware performed better than sequential for less number of nodes whereas worse for a large number of nodes.

Conclusions and Future Work

- Our algorithm reduces run-times by more than 38% over random, sequential and load-aware allocations.
- Formalization of weights estimation
- Extension to large scale systems, spanning over multiple clusters.

INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING
ICPP/2020/EDMONTON/CANADA
AUGUST 17-20, 2020

Developing Checkpointing and Recovery Procedures with the Storage Services of Amazon Web Services

Luan Teylo[1], Rafaela C. Brum[1], Luciana Arantes[2], Pierre Sens[2] & Lúcia M. A. Drummond[1]

Federal Fluminense University - IC/UFF[1], Sorbonne Université - LIP6[2]

Motivation

Clouds, usually, offer VMs in different markets, with different guarantees in terms of availability and prices

- On-demand VMs:
  - High availability
  - Cannot be interrupted by the provider
- Spot VMs:
  - Offer up to 90% discount compared with on-demand prices
  - Low availability
  - Interrupted by the provider when it needs the resources back

As the VMs in the **spot market** are subject to revocation by the provider, the adoption of checkpoint/recovery techniques are essential to minimize possible job losses

When using a checkpoint, it is essential to ensure that, in the event of an interruption, the files required for the task recovery are available. In the case of cloud environments, different storage options can be hired and used along with the VMs.

This work proposes and evaluates checkpoint and recovery procedures by adopting the following storage services:

- Amazon Simple Storage Service (S3), an object storage service that offers scalability, security and performance;
- Amazon Elastic Block Store (EBS), a block storage service designed for EC2 VMs and workloads with high throughput;
- Amazon Elastic File System (EFS), a simple and scalable elastic NFS file system.

Contributions

The checkpoint/recovery procedures were included into a previously proposed framework, called HADS (Hibernation Aware Dynamic Scheduling), for scheduling bag-of-tasks (BoT) applications onto the spot and on-demand VMs, aiming at minimizing monetary costs and respecting a given deadline.
The main contributions of this paper are the following:

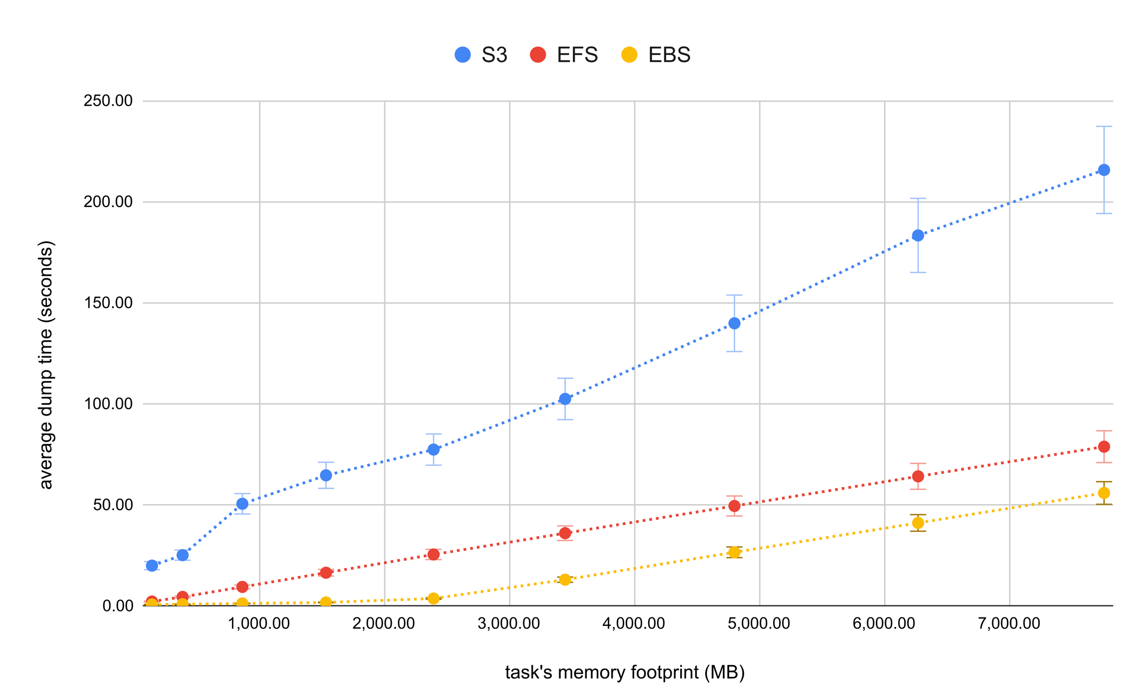
- Extension of HADS with new checkpoint and recovery procedures;
- Evaluation of the scalability and impact of the proposed strategies in terms of execution and monetary costs, in different storage services.

Results

Dump time without concurrence

The dump time with S3 presented an increment of **72.57%** and **89.37%** on average when compared to EFS and EBS, respectively

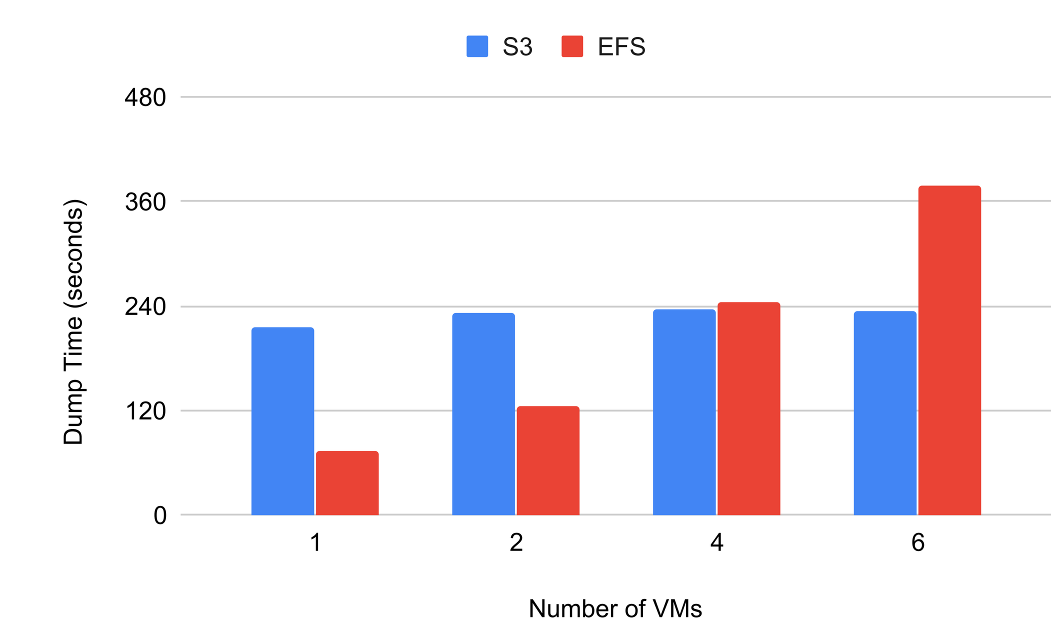
EBS presented the best results, with dump time varying from 0.65 to 55.82 seconds, followed by EFS (2.12 to 78.73 seconds)



Dump time with concurrence

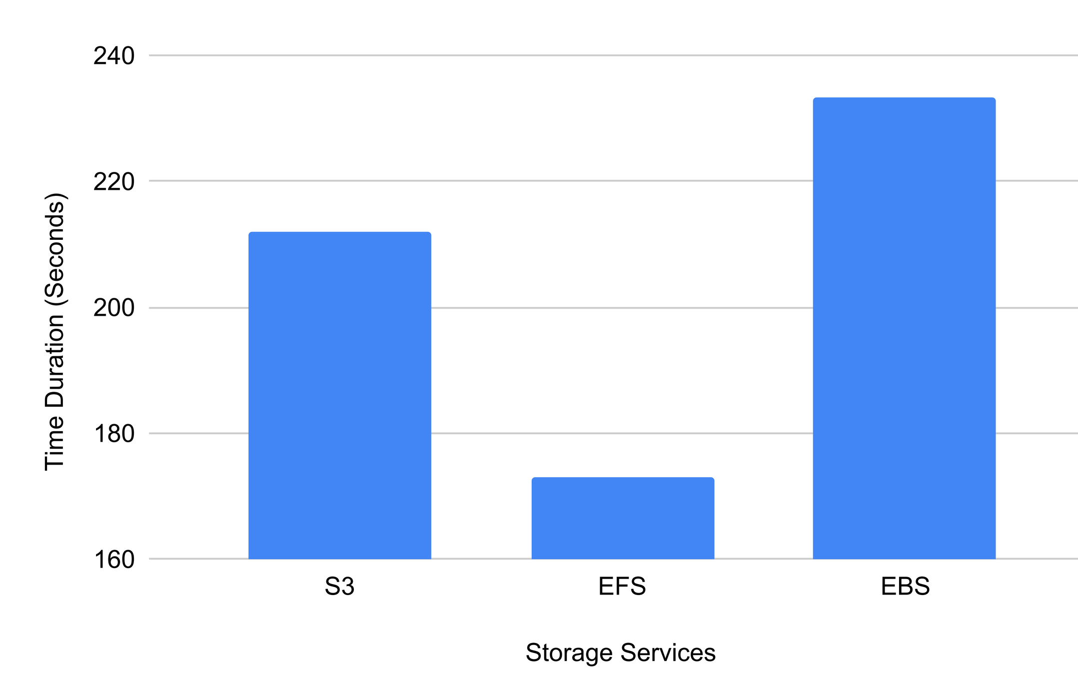
Task with the biggest memory footprint (7,750 MB) was executed considering scenarios where one, two, four, and six VMs shared the same file system. To avoid concurrency with other resources, we considered only one task per VM
The average dump time with S3 was 65.92% greater than EFS with one VM. That difference drops to 46.31% with two VMs. at the four VMs scenario, the time already becomes bigger in EFS then S3 (3.03% of increment)

In the six VMs scenario, the dump time with concurrent checkpoint recording increased **37.89%** with EFS in comparison to S3.



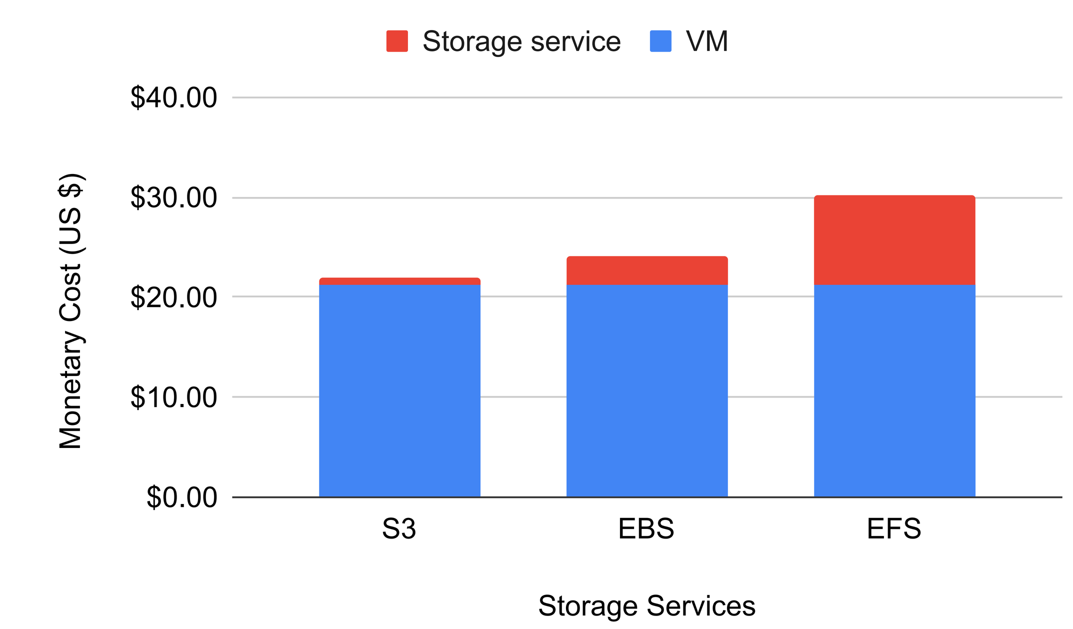
Recovery Procedure Evaluation

The time of EBS is **9.14%** higher than S3 and **25.86%** higher than EFS



Monetary Cost for Long-Running Tasks

We considered an application with only one task executing for 30 days without any interruption or revocation. We assumed that 30 GBs of data were kept in the storage service, including the checkpoint files, along those days.
While the user pays **US$0.69** for the 30 GBs stored for 30 days in S3, in EBS and EFS those costs are **US$3.0** and **US$9.01**, respectively



Conclusion & Future Work

S3 proved to be the best option in terms of monetary cost but required a longer time for recording a checkpoint, individually. However, when concurrent checkpoints were analyzed, which can occur in a real application with lots of tasks, in our tests, S3 outperformed EFS in terms of execution time also

Next Steps:

- We intend to evaluate other checkpoint approaches, including the two-step asynchronous recording
- The impact of the used checkpoint interval on the monetary cost and execution time

Contact Information

- Lab: cloud.ic.uff.br
- luanteylo@id.uff.br