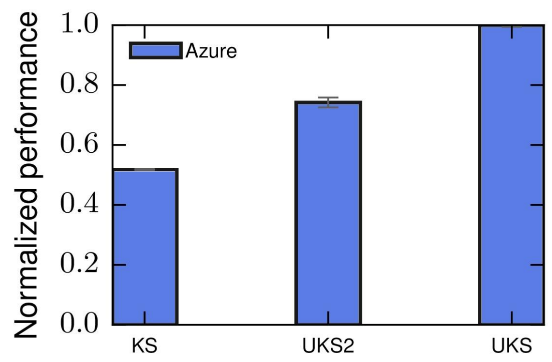
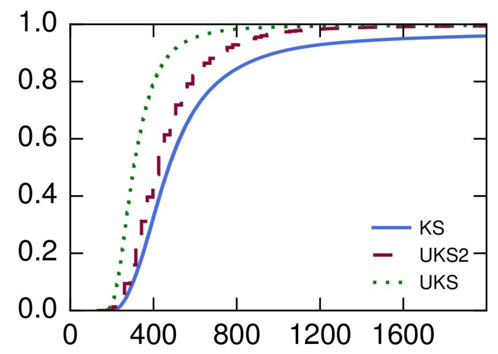
Latency-Driven, Application Performance-Aware, Cluster Scheduling

Diana Andreea Popescu, Andrew W. Moore
University of Cambridge
Contact: diana.popescu@cl.cam.ac.uk

Motivation

- Network latency variability is common in multi-tenant data centers, leading to performance variability [1,3]. Even small amounts of delay, in the order of microseconds, may lead to significant drops in application performance [1].
- For example, we obtained different performance values for Memcached in different data centres, and in the same data centre at different times after restarting the VMs.
- We place the applications according to how latency-sensitive they are, and to the current measured latency in the data centre, which is not constant [13]. If latency increases, the application may be migrated.



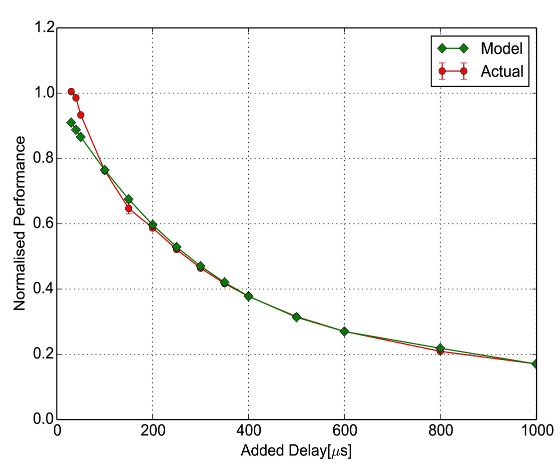


Modeling Application Performance

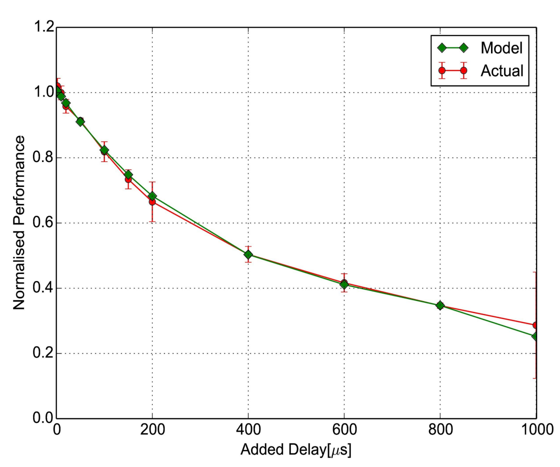
- We studied the effect of network latency on application performance, as defined for a certain application.
- We did this by artificially injecting arbitrary network latency into a networked system using a bespoke hardware appliance [1,2].
- We fit a curve to the observed results to find *p(injected latency)= normalized application performance metric,* where *p* is the performance.
- For the small latency values the model can be assimilated to a constant function whose value is the baseline performance.

Application	Role	#Hosts	Metric	Runtime Target	Dataset	Dataset Size
Memcached [4]	Server	5	Queries/sec	10 seconds	FB ETC [5]	See [5]
Tensorflow Handwritten digit recognition [6]	Server	9	Training time	20K iterations	MNIST	60K examples
STRADS [7] Lasso Regression	Coordinator	6	Training time	100K iterations	Synthetic	10K samples, 100K features
Spark [8] Ridge Regression	Master	8	Training time	100 iterations	Spark-perf generator[8]	100K samples, 10K features

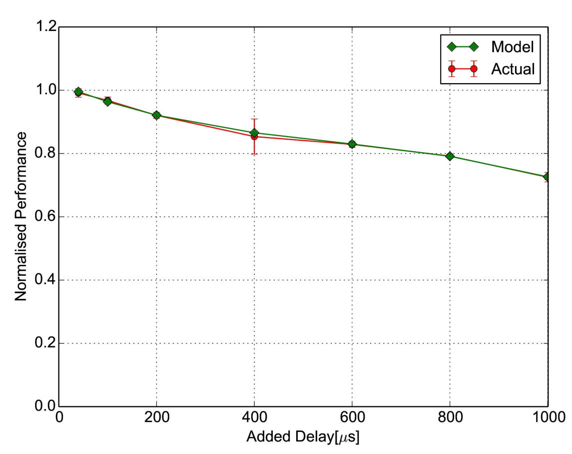
Memcached



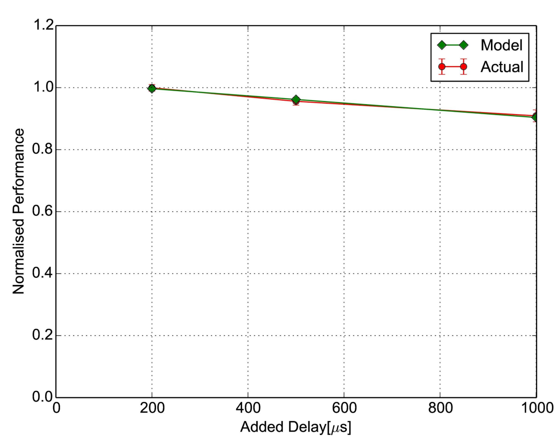
STRADS



Tensorflow



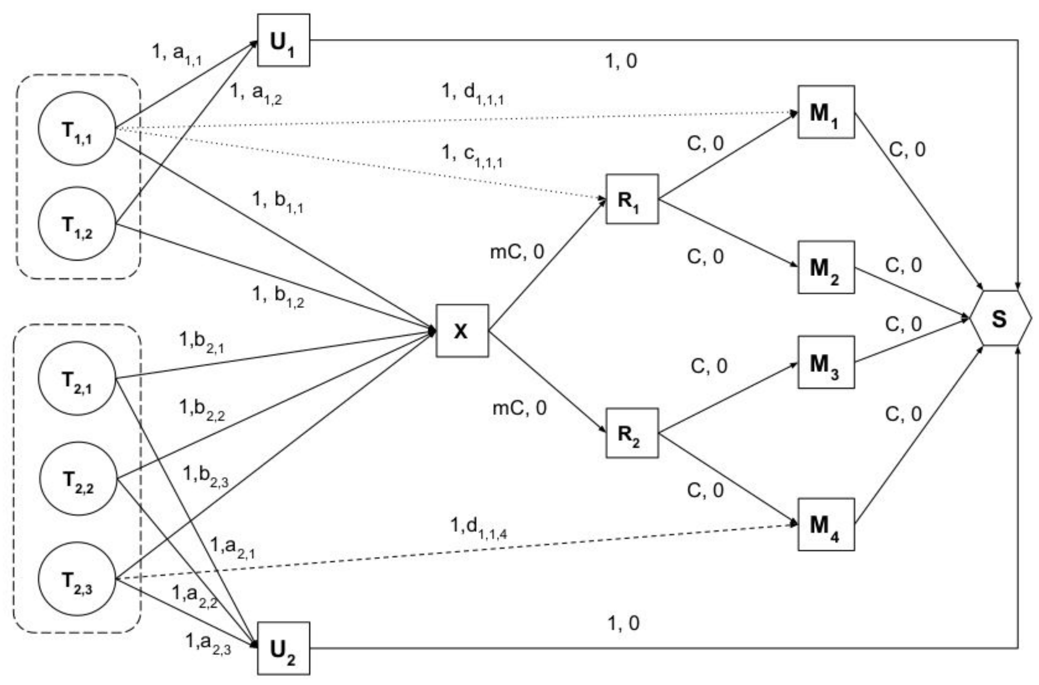
Spark



This work was supported by the EU FP7 METRICS ITN (grant number 607728), EU Horizon 2020 research and innovation programme 2014-2018 under the SSICLOPS (grant agreement No. 644866).

NoMora Cluster Scheduling Policy

- NoMora architecture:
  - Functions that predict application performance dependent upon network latency;
  - Network latency measurement system (Pingmesh [9], PTPmesh[10]);
  - **Latency-driven, application performance-aware, cluster scheduling policy** implemented on top of the Firmament [11] cluster scheduler, which models the cluster scheduling problem as a max-flow min-cost problem.
- Flow network: T - task of a job , R - rack, M - machine(host), X - cluster aggregator, U - unscheduled aggregator, S - sink, C - number of cores on a machine; a, b, c, d costs on arcs
- Jobs: have a root task (the server/the master and the clients/workers)
- Placement algorithm:
  - the root task is scheduled on any available machine (the root task is assigned a single arc to the cluster aggregator, with a cost of 0);
  - if a task that is not a root task enters the system at the same time as the root task, or before the root task is scheduled, it will not be scheduled, waiting instead;
  - if the root task is scheduled, then a new task's placement is determined based on the application performance prediction, and current network latencies to the root task's placement.



$$d_{i,j,m} = cost(T_{i,j}, M_m) = \frac{1}{p(max(latency(M_{root}, M_m)))} \qquad c_{i,j,r} = cost(T_{i,j}, R_r) = \max_{m \in r} \frac{1}{p(max(latency(M_{root}, M_m)))}$$

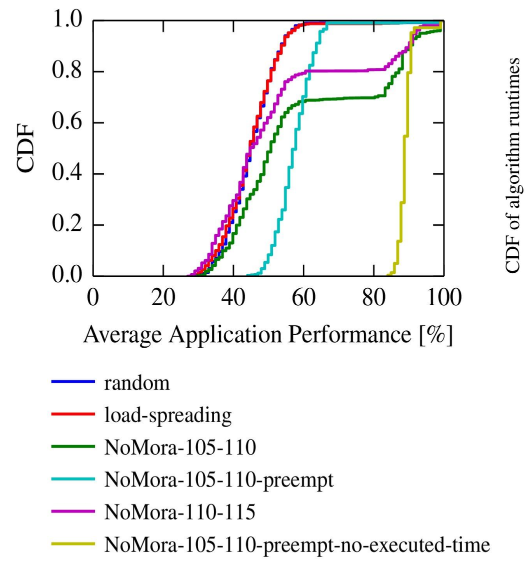
$$a_{i,j} = \omega \times \alpha_{i,j} + \gamma$$

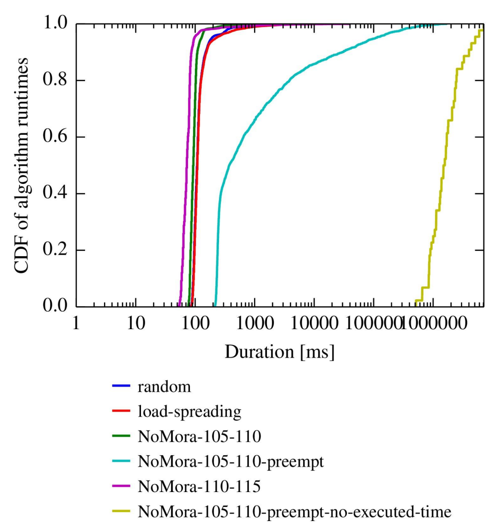
$\alpha_{i,j}$ task wait time

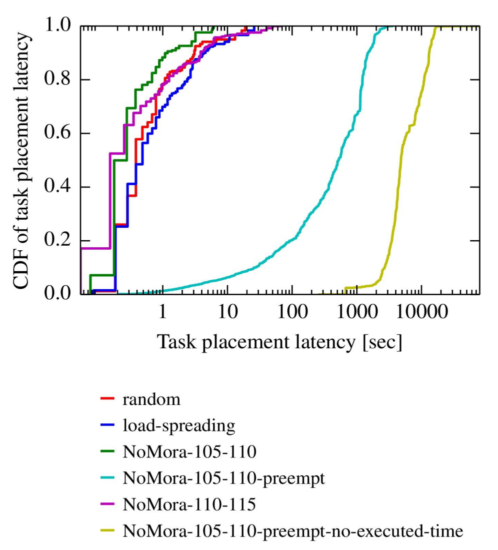
$$b_{i,j} = \max_r c_{i,j,r}$$

NoMora Evaluation

- Simulation setup:
  - Google cluster trace [12]
  - Network latency measurements from [13]
  - Topology - number of hosts per rack 16, number of racks per pod 48
- Evaluation metrics:
  - **Average application performance**: measures task placement quality;
  - Algorithm runtime;
  - Task placement latency.
- Average application performance improves by up to 13.4% and by up to 42% if migration is enabled, compared to the baselines.
- The task placement latency improves by a factor of 1.79× and the median algorithm runtime by 1.16× compared to the baselines.







[1] *Characterizing the impact of network latency on cloud-based applications' performance*, Diana Andreea Popescu et al., Technical Report, Number 914, UCAM-CL-TR-914, ISSN 1476-2986, November 2017,University of Cambridge, UK
[2] *Where Has My Time Gone?,* Noa Zilberman, Matthew Grosvenor, Diana Andreea Popescu, Neelakandan Manihatty-Bojan, Gianni Antichi, Marcin Wojcik, Andrew W. Moore*, PAM 2017*
[3] *Inferring the Network Latency Requirements of Cloud Tenants*, Jeffrey C. Mogul and Ramana Rao Kompella, HotOS 2015
[4] *Mutilate,* https://github.com/leverich/mutilate
[5] *Workload Analysis of a Large-scale Key-value Store*, Atikoglu and et al., *ACM SIGMETRICS 2012*
[6] *Tensorflow,* https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/tutorials/mnist
[7] *STRADS: A Distributed Framework for Scheduled Model Parallel Machine Learning,*Kim et al., *EuroSys 2016*
[8] *Spark,* https://github.com/databricks/spark-perf
[9] *Pingmesh: A Large-Scale System for Data Center Network Latency Measurement and Analysis,* Guo et al., *ACM SIGCOMM 2015*
[10] *PTPmesh: Data center network latency measurements using PTP,* Diana Andreea Popescu and Andrew W. Moore, *IEEE MASCOTS 2017*
[11] *Firmament: Fast, Centralized Cluster Scheduling at Scale,* Gog et al., *OSDI 2016*
[12] https://github.com/google/cluster-data
[13] *A First Look at Data Center Network Conditions Through The Eyes of PTPmesh,* Diana Andreea Popescu and Andrew W. Moore, *IFIP/IEEE TMA 2018*