# Bandwidth-Aware Placement in Hadoop Clusters

XXX
*xxx@illinois.edu*

## Abstract

Hadoop and its variants have been highly successful in implementing large-scale data-intensive applications on commodity clusters. However, these systems have not considered the bandwidth availability during task-scheduling. The advent of Software-Defined Networking (SDN) techniques has enabled in choosing the best routing path but the responsibility of machines' placement lies within Hadoop. In this paper, we propose a bandwidth-aware placement scheme in Hadoop that could also be used to complement SDN techniques. Our algorithm improves the throughput of Hadoop applications by 20% in Fat-Tree [4] and Jellyfish [5] topology.

## 1 Design

@NOTE: Try linear programming approach as discussed in VM placement papers?

The problem of optimizing throughput of Hadoop applications are inherently two-folds; deciding the placement of mapper and reducer jobs and then, deciding the routing between the mappers and reducers.

@TODO: When to move job around?. @TODO: Apps might have min. bandwidth requirement

To this end, our algorithm use two levels of Simulated Annealing (SA), one for placement decisions and another for routing decisions. We also could substitute the second-level of SA with other approaches in SDN [3].The first-level of SA loops to explore the optimal placement of mappers and reducers placement while the second-level of SA loops to explore the optimal routing paths of the mappers and reducers. To help in the routing paths computation, we modify Floyd-Warshall Algorithm [1] to compute all-pairs k-shortest paths instead of just the shortest path and the results are then cached.

When a new job arrives, it first specifies the number of mappers, $M$, and the number of reducers, $R$. Since Hadoop keep tracks of the nodes that are currently free, $freeHost$, the first-level of SA will then iterate through $\binom{freeHost}{(M+R)}$ possible states. In each iteration, the first-level of SA will then call the second-level of SA. From our pre-computed routing paths, each mapper has k paths to communicate with each reducer. Thus, our second-level of SA will then iterate through $k^2$ possible states. In each iteration, the second-level of SA will compute the max-min fairness of the currently running jobs in conjunction with the potential paths and placements of the new job.

**Algorithm 1** Simulated Annealing Algorithm

```
1:  currentState ← initState()
2:  for i ← 1 to maxStep do
3:      newState ← genState(currentState)
4:      newUtil ← computeUtil(newState)
5:
6:      if transition(currentUtil, newUtil) then
7:          currentState ← newState
8:          currentUtil ← newUtil
9:      end if
10:
11:     if currentUtil ≥ bestUtil then
12:         bestUtil ← currentUtil
13:         bestState ← currentState
14:     end if
15: end for
16: return bestState
```

## 2 Evaluation

We build a map-reduce simulator that evaluates the performance of our bandwidth-aware placement algorithm using Facebook workload in 2009 provided by the SWIM benchmark [2] under multiple network topology; Fat-Tree, Jellyfish and a slight modification to Jellyfish in

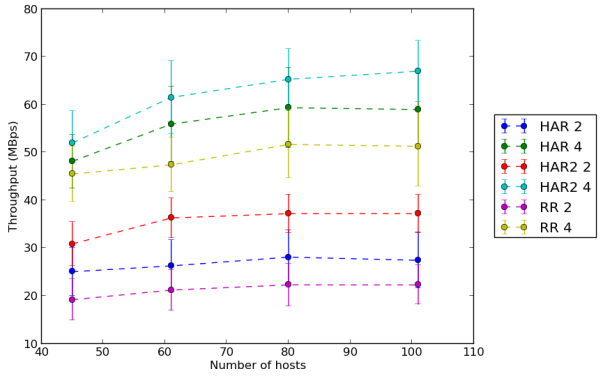


Figure 1: Throughput in Jellyfish2 with varied routing strategies

which with 50% and 25% probability, a node will be connected to 1 and 3 additional switches respectively which will be referred to from this point as Jellyfish2.

In each of the graphs below will be labeled in the format $< routingStrategy >< numMR >$. $< routingStrategy >$ might be one of three things; RR indicates random routing (i.e. both placements and routings are being done at random), HAR indicates simulated annealing for routing decisions, while HAR2 indicates simulated annealing for both placement and routing decisions. $< numMR >$ indicates the number of mappers and reducers that each job will request in the beginning of their execution.

All of the experiments are run on a 4-core 2.50 GHz processor. The bandwidth of each link is set to 100Mbps. Section 2.1 evaluates the throughput of jobs using our algorithm while section 2.2 evaluates the jobs' completion time. Since we have two levels of SA, we will also show the benefits provided by each level.

## 2.1 Throughput

In Jellyfish and Fat-Tree, the throughput improvement provided by SA for routing decisions is 11% and for routing and placement decisions is 14% when the number of hosts is 45 as opposed to random routing. The improvement decreases to almost 0 when we set the number of hosts to 100. This is due to the fact that the throughput is bottlenecked by the link connecting host nodes with the switch. For instance, if the number of mappers and reducers are each set to 2, the upper-limit in normal Jellyfish and Fat-Tree topology will only be $2 * linkBandwidth$ while optimally, it should be $2 * 2 * linkBandwidth$. As the graph shows, both one level and two levels of SA quickly hits the upper-bound on the total amount bandwidth of the graph.

However, the throughput improvements are signifi-

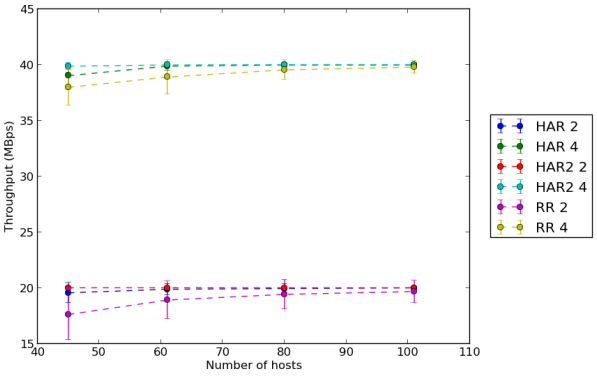


Figure 2: Throughput in Jellyfish with varied routing strategies

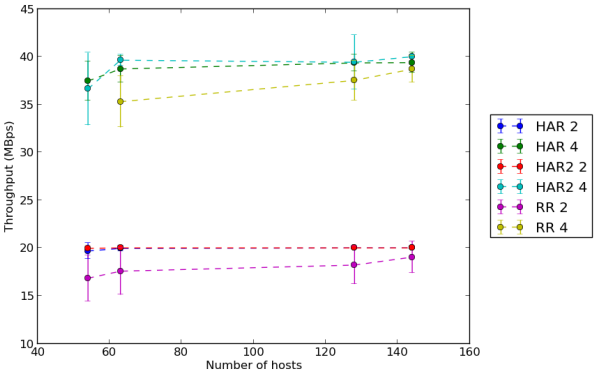


Figure 3: Throughput in Fat-Tree with varied routing strategies

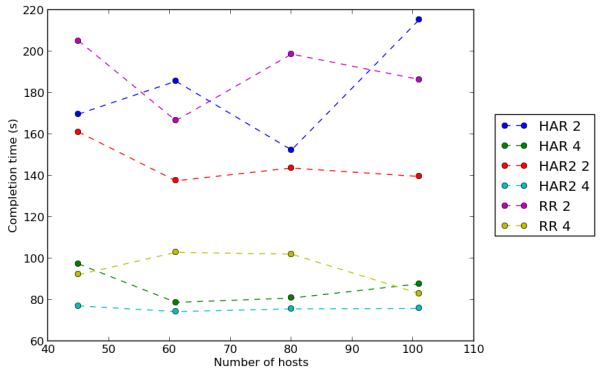


Figure 4: Jobs completion time in Jellyfish2 with varied routing strategies

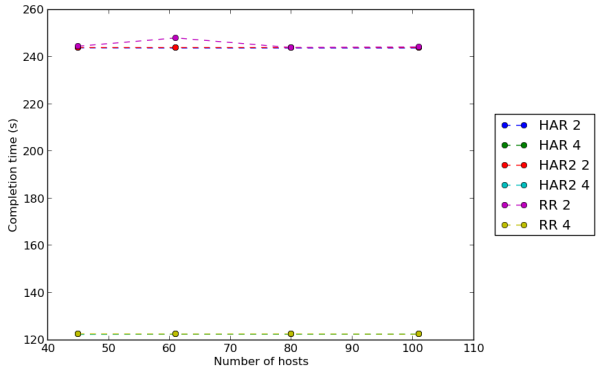


Figure 5: Jobs completion time in Jellyfish with varied routing strategies

cantly better in Jellyfish2 since the aforementioned bottleneck is reduced as some nodes could be directly connected to as many as 4 switches instead of just one. As shown in the graph, the throughput improvement provided by each level of SA is constant (15%) as we scale the number of hosts;

@NOTE: Thought: Start thinking about how data is being moved around as opposed to just focusing on shuffle phase?

@NOTE: Thought: Evaluate with varying map-reduce tasks?

## 2.2 Completion Time

@TODO: Separate into bins?

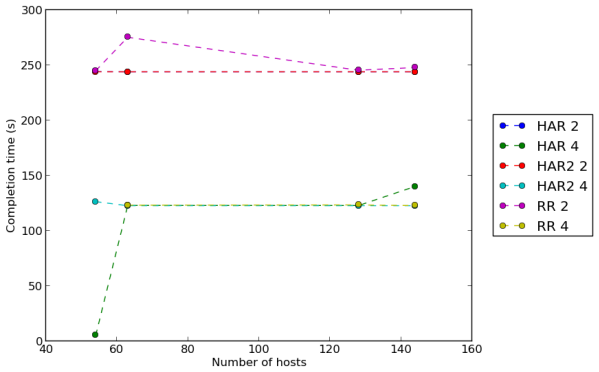


Figure 6: Jobs completion time in Fat-Tree with varied routing strategies

## References

[1] Floyd-warshall algorithm, Aug. 2013.

[2] Statistical workload injector for mapreduce (swim), Aug. 2013.

[3] A. Das, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and C. Yu. Transparent and flexible network management for big data processing in the cloud.

[4] C. E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, Oct. 1985.

[5] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers randomly. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 17–17. USENIX Association, 2012.