

Scheduling a Large DataCenter

Cliff Stein
Columbia University
Google Research

Monika Henzinger, Ana Radovanovic
Google Research, U. Vienna



Scheduling a DataCenter

- Companies run large datacenters
- Construction, maintenance, etc. of datacenters has significant cost, and uses a significant amount of power
- Datacenters are estimated to use 2% of the power in the United States
- Managing such a data center efficiently is an important problem
- We will talk about experience scheduling google's datacenters

Power concerns

What matters most to the computer designers at Google is not speed, but power, low power, because data centers can consume as much electricity as a city

Eric Schmidt, CEO Google

Energy costs at data centers are comparable to the cost for hardware



An abstraction of a computing environment

- Users submit **jobs** consisting of **tasks**.
- Tasks are the unit that is scheduled.
- Mix of long-running and short-running jobs.
- Mix of user-facing and back-end jobs.
- Mix of high and low priority jobs.

- We will consider a “datacenter” with thousands of (heterogeneous) machines, and a time period (“day”) long enough to have hundreds of thousands of tasks.

The goal

- We want to evaluate the performance of many different scheduling algorithms on a large datacenter, make meaningful comparisons and recommend better algorithms
- **High Level Goal:** *improve cells utilization, overall productivity, energy useage*

Meta-goal

- How does one actually carry out such an experiment?



Some ways to measure scheduling quality

- *Throughput* - number of processed tasks
- *Total flow time* – total time tasks spend in system
- *Total useful work* – total time tasks spend processing work that will not be thrown away
- *Number of preemptions* – times tasks were interrupted.
- *Pending queue size* – number of tasks in system but not being scheduled
- *Machine fragmentation* – roughly the number of unused machines

Primary Goals

- **Increase throughput.**
- **Reduce machine fragmentation** (increase job packing ability).
- **Increase the number of unused machines** (leads to power savings).

Overview

- We collected data from google datacenters
- We built a high-level model of the scheduling system (how do you figure this out?)
- We experimented with various algorithms

How to model machines and jobs


➤ Machines:

- Disk
- Memory
- CPU

➤ Jobs

- Consist of set of tasks, which have
 - Cpu, disk, memory, precedence, priority, etc.
 - Processing times (**how do you compute these?**)
 - Long list of other possible constraints

Simulator

- Replay a “day” of scheduling using a different algorithm.
 - Use data gathered from checkpoint files kept by the scheduling system
 - We tried 11 different algorithms in the simulator.
- 

The Algorithmic Guts of Scheduling

Given a task, we need to choose a machine:

1. Filter out the set of machines it can run on
2. Compute $\text{score}(i,j)$ for task j on each remaining machine i .
3. Assign task to lowest scoring machine.

Notes:

- The multidimensional nature of fitting a job on a machine makes the scoring problem challenging.

Algorithms

If we place task j on machine i , then

- $\text{free_ram_pct}(i) =$
free ram on i (after scheduling j) / total ram on i
- $\text{free_cpu_pct}(i) =$
free cpu on i (after scheduling j) / total cpu on i
- $\text{free_disk_pct}(i) =$
free disk on i (after scheduling j) / total disk on i

Algorithms

- **Bestfit:** Place job on machine with “smallest available hole”
 - V1: $\text{score}(i,j) = \text{free_ram_pct}(i) + \text{free_cpu_pct}(i)$
 - V2: $\text{score}(i,j) = \text{free_ram_pct}(i)^2 + \text{free_cpu_pct}(i)^2$
 - V3: $\text{score}(i,j) = 10 \text{ free_ram_pct}(i) + 10 \text{ free_cpu_pct}(i)$
 - V4: $\text{score}(i,j) = 10 \text{ free_ram_pct}(i) + 10 \text{ free_cpu_pct}(i) + 10 \text{ free_disk_pct}(i)$
 - V5: $\text{score}(i,j) = \max(\text{free_ram_pct}(i), \text{free_cpu_pct}(i))$
- **Firstfit:** Place job on first machine with a large enough hole
 - V1: $\text{score}(i,j) = \text{machine_uid}$
 - V2: $\text{score}(i,j) = \text{random}(i)$ (chosen once, independent of j)
- **Sum-Of-Squares:** tries to create a diverse set of free machines (see next slide)
- **Worst Fit (EPVM):** $\text{score}(i,j) =$
 - $-(10 \text{ free_ram_pct}(i) + 10 \text{ free_cpu_pct}(i) + 10 \text{ free_disk_pct}(i))$
- **Random:** Random placement

Sum of Squares

Motivation: create a diverse profile of free resources

- Characterize each machine by the amount of free resources it has (ram, disk, cpu).
- Define buckets: each bucket contains all machines with similar amounts of free resources (in absolute, not relative size).
- Let $b(k)$ be the number of machines in bucket k .
- $\text{Score}(i,j) = \sum b(k)^2$ (where buckets are updated after placing job j on machine i).
- Intuition: function is minimized when buckets are equal-sized.
- Has nice theoretical properties for bin packing with discrete sized item distributions.

Two versions:

- V1: bucket ram and cpu in 10 parts, disk in 5 = 500 buckets.
- V2: bucket ram and cpu in 20 parts, disk in 5 = 2000 buckets.

Sum of Squares (1-D)

- Suppose four machines with 1G of Ram:
 - M1 is using 0G
 - M2 is using 0G
 - M3 is using .25G
 - M4 is using .75G
- Bucket size = .33G. Vector of bucket values = (3,0,1). $\sum b(k)^2 = 10$.
- .5G job arrives.
 - If we add a .5G job to M1 or M2, vector is (2,1,1). $\sum b(k)^2 = 6$.
 - If we add a .5G job to M3, vector is (2,0,2). $\sum b(k)^2 = 8$.
- We run the job on M1.
- This algorithm requires more data structures and careful coding than others.

Algorithm Evaluation

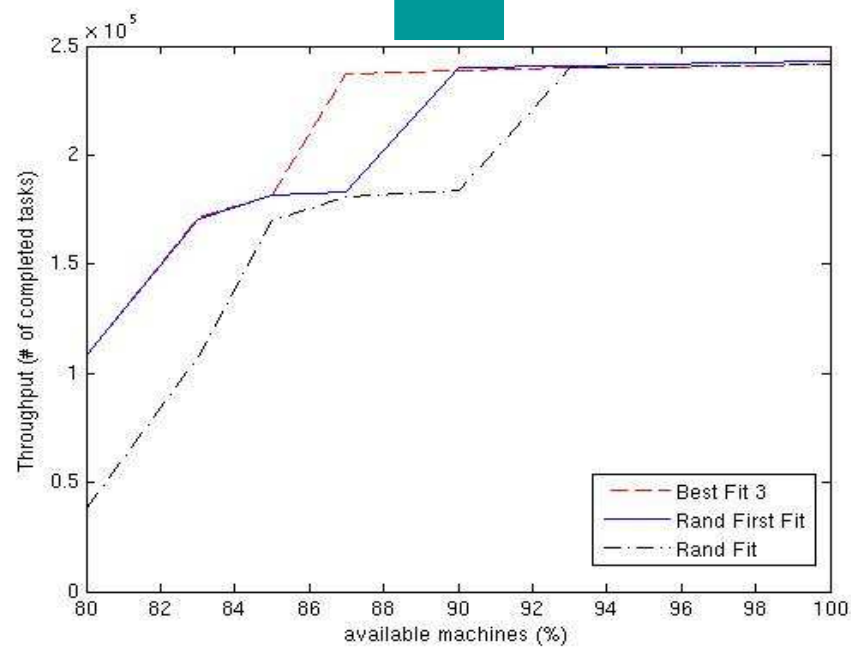
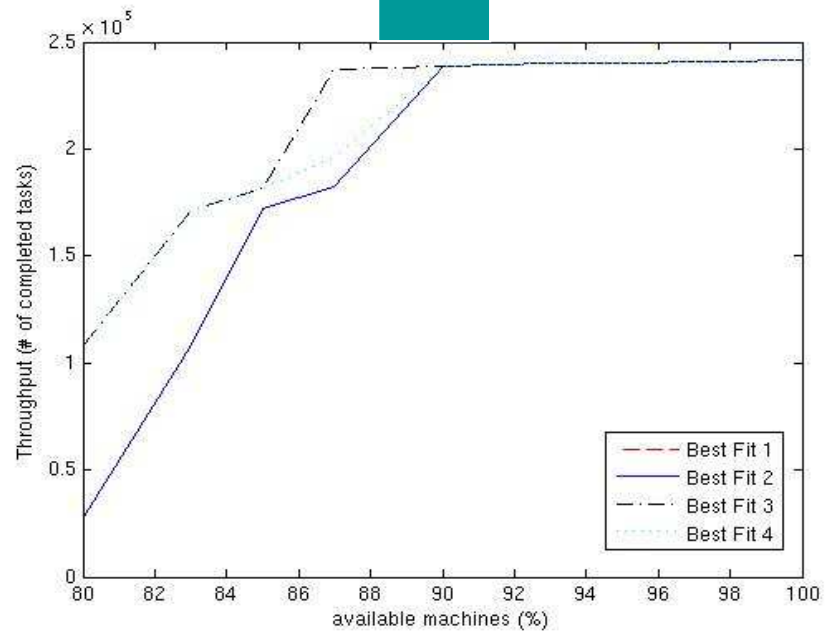
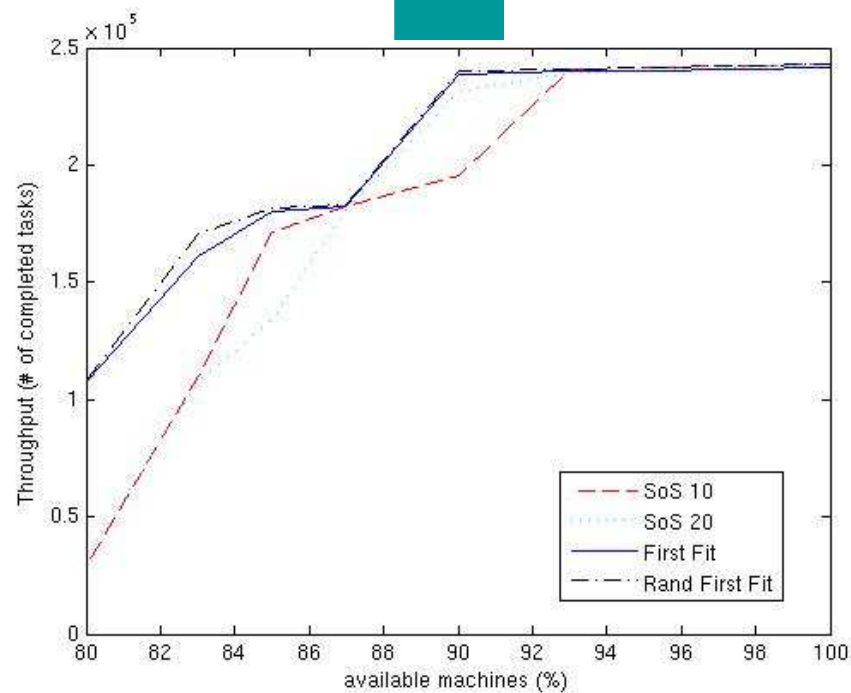
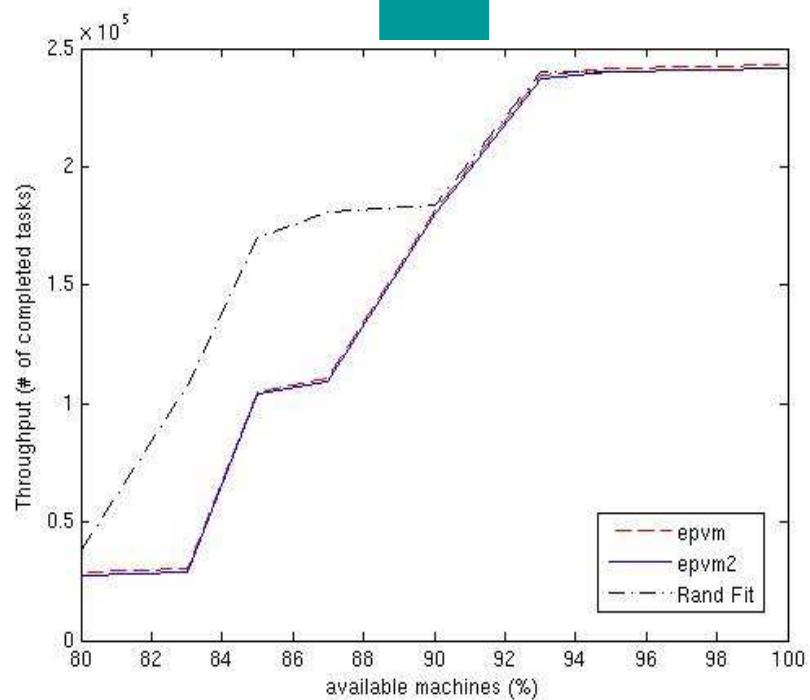
Big Problem:

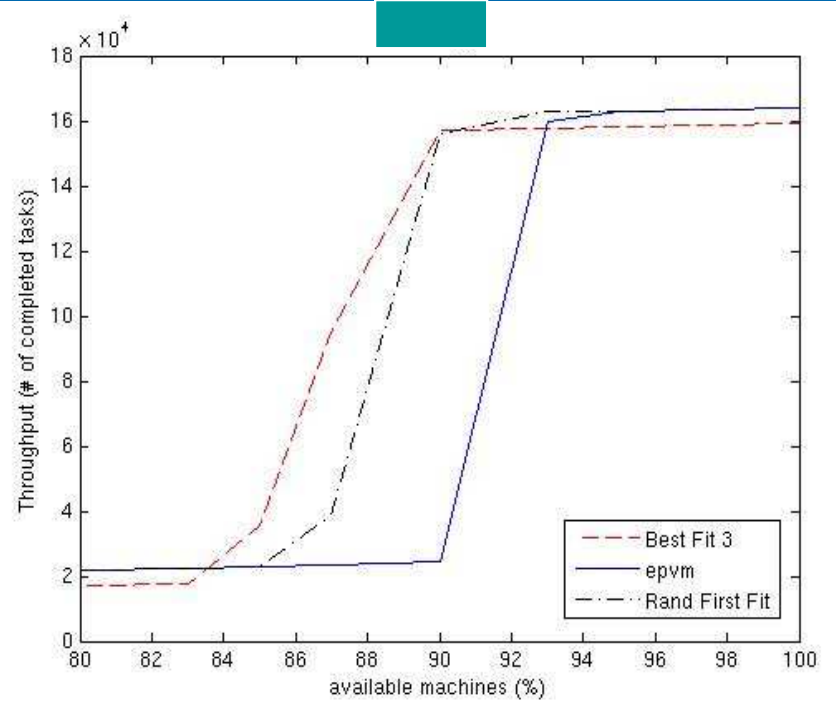
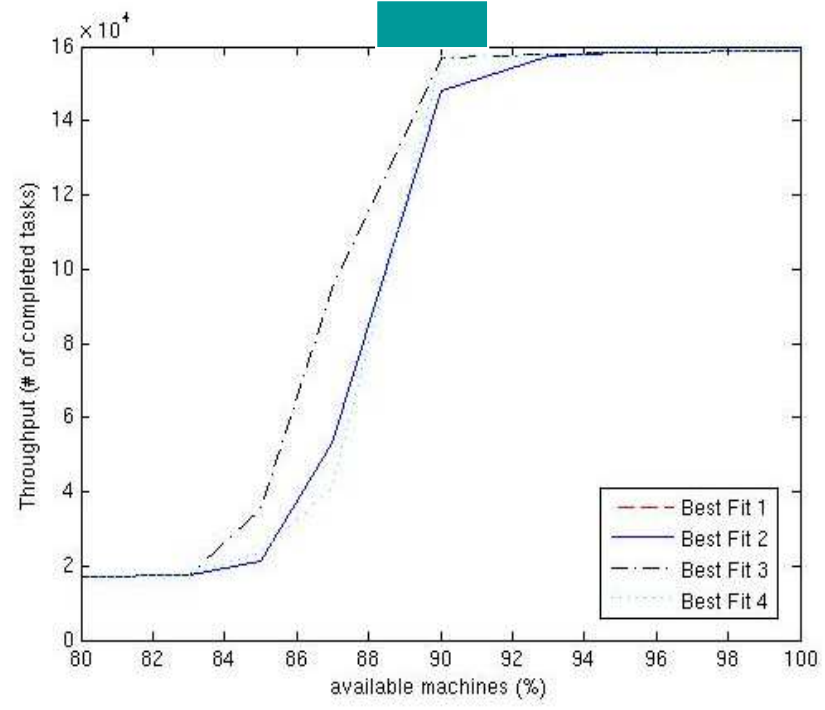
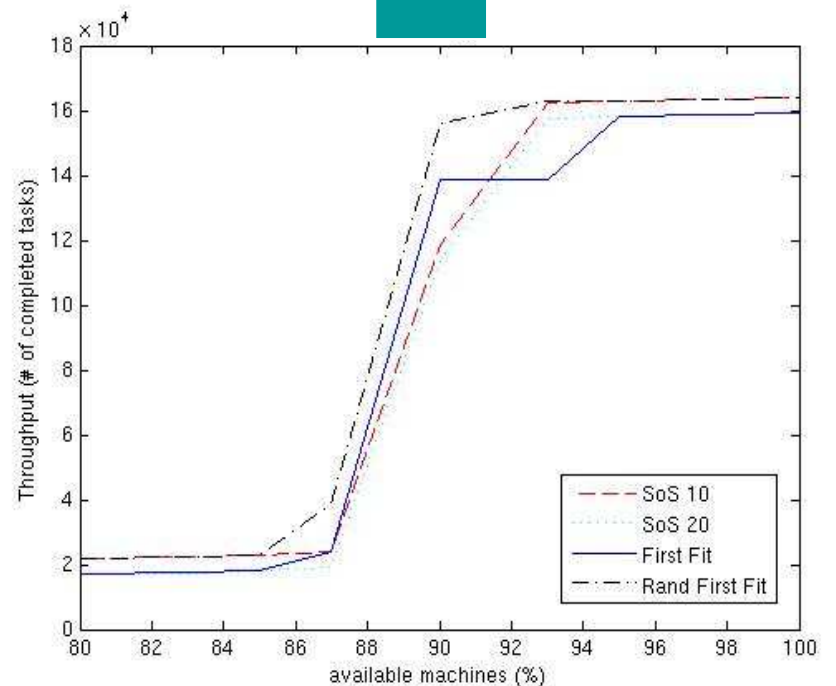
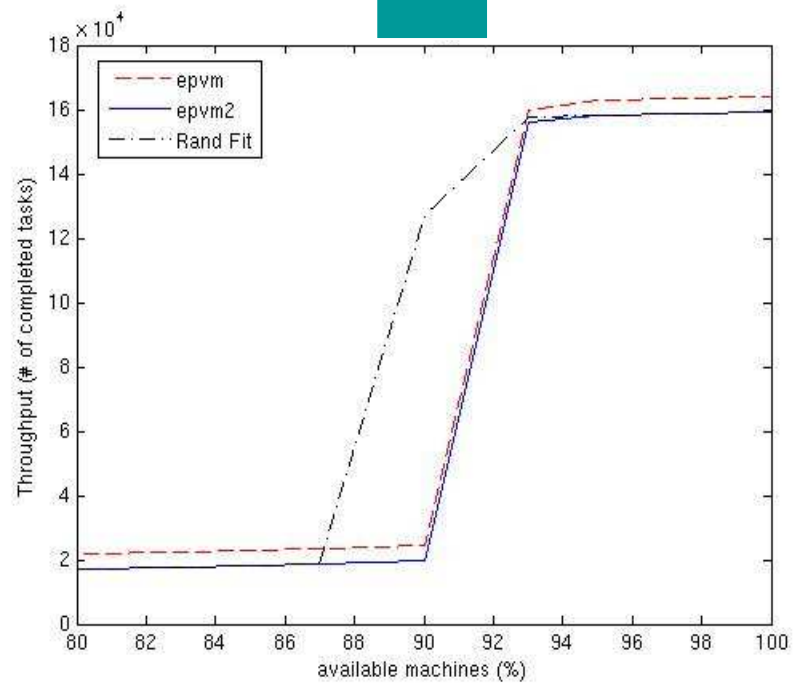
- If a cell ran all its jobs and is underloaded, almost any algorithm is going to do reasonably well.
- If a cell was very overloaded and didn't run some jobs, we might not know how much work was associated with jobs that didn't run.

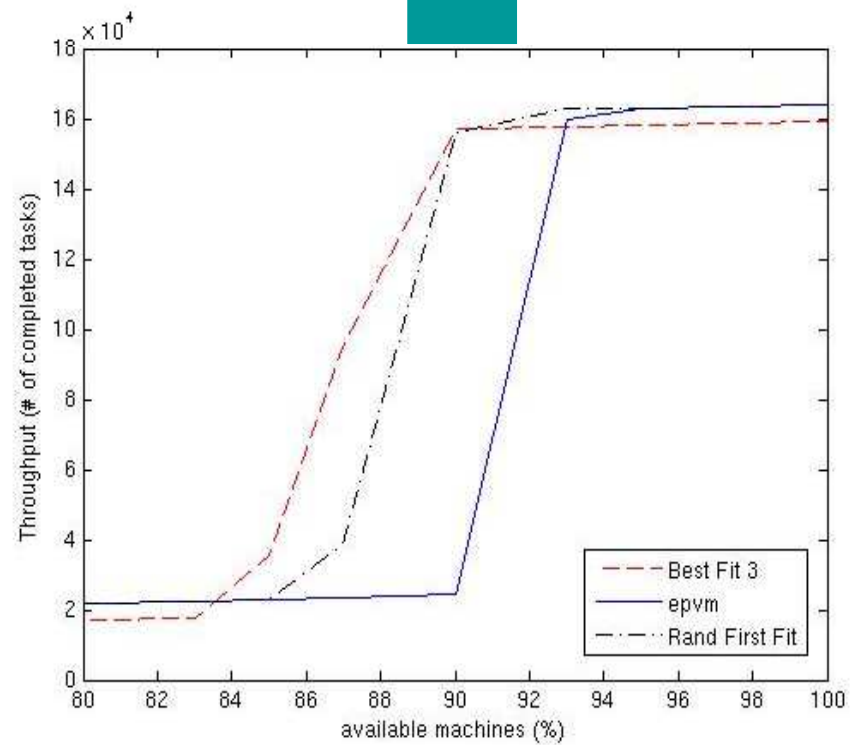
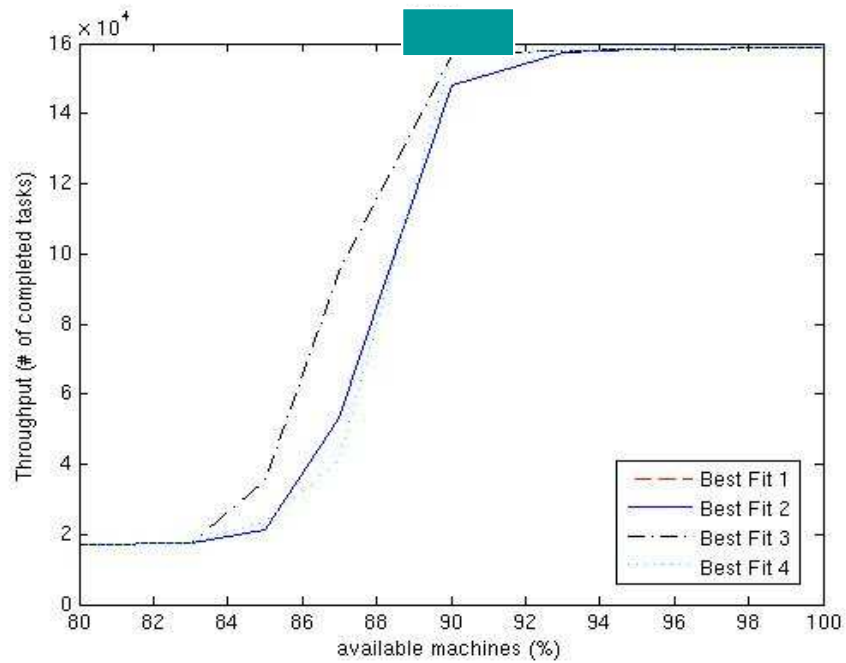
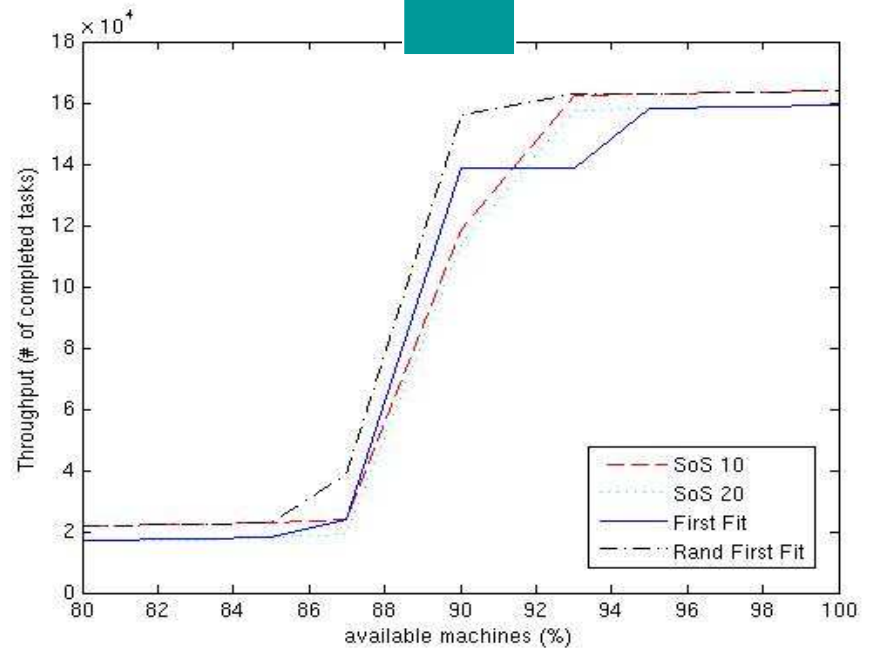
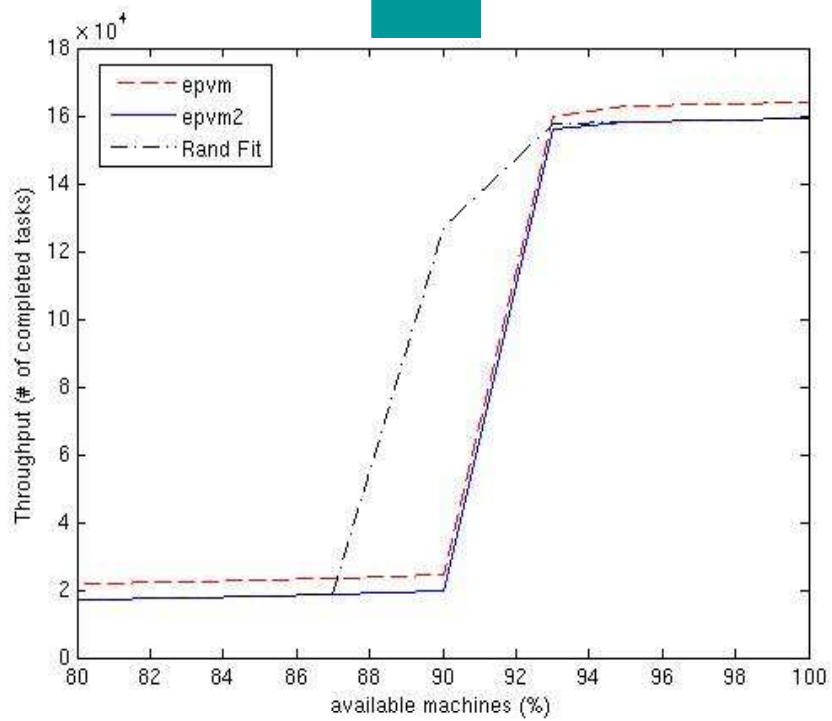
Algorithm Evaluation Framework

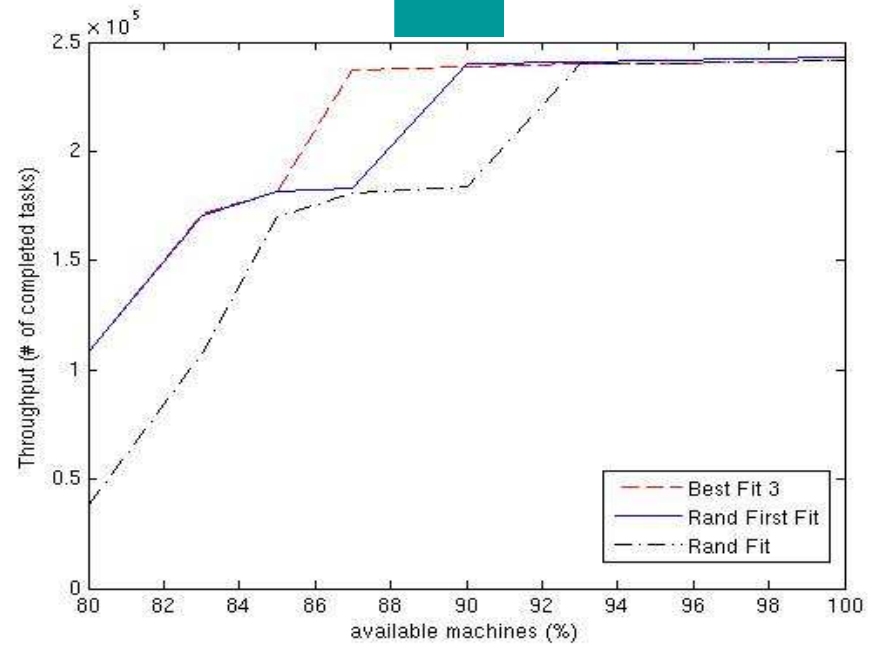
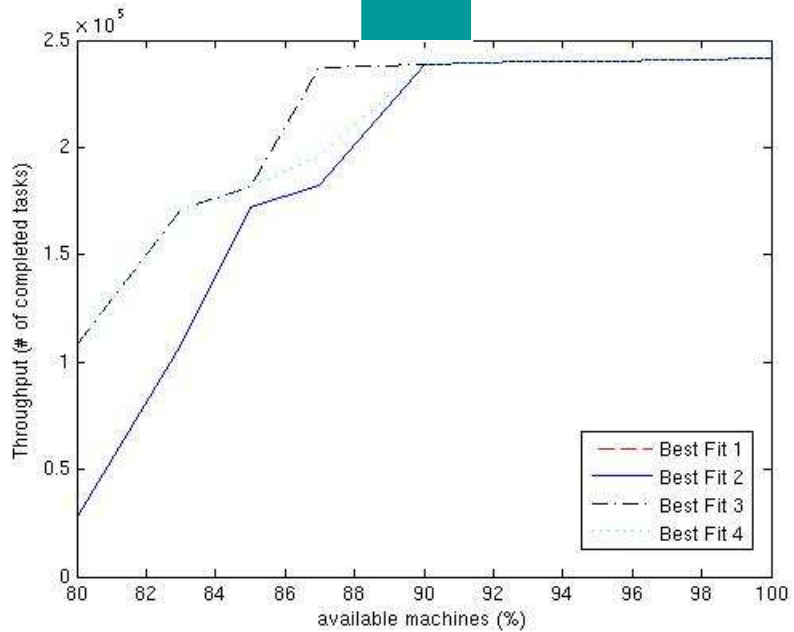
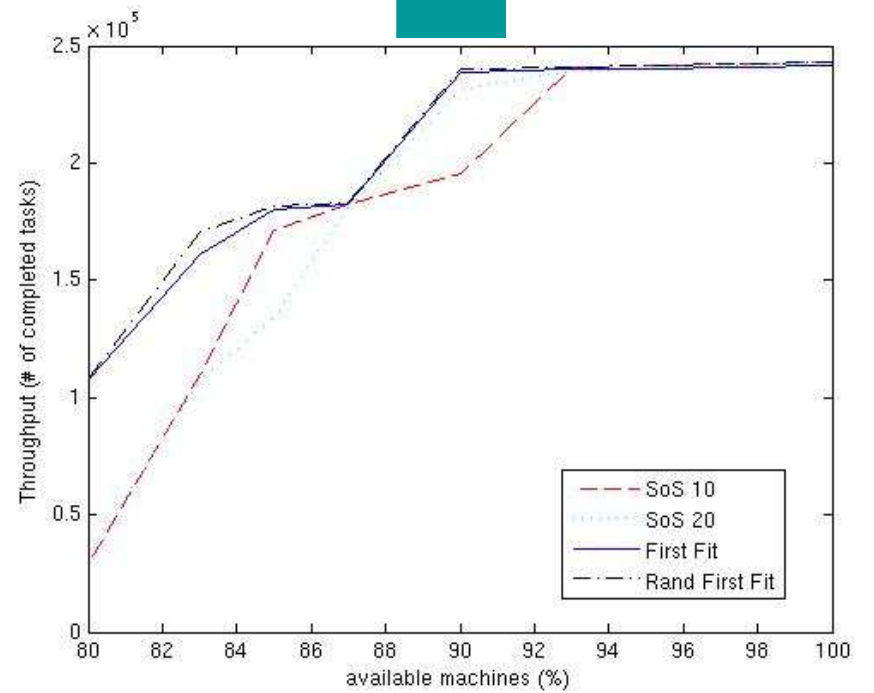
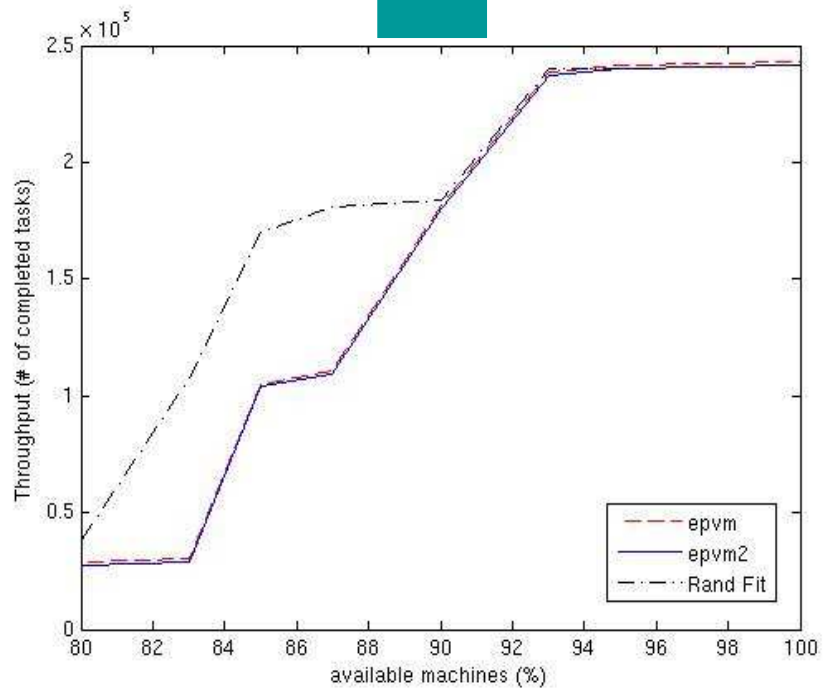
As an example, let's use the metric of throughput (number of completed jobs).

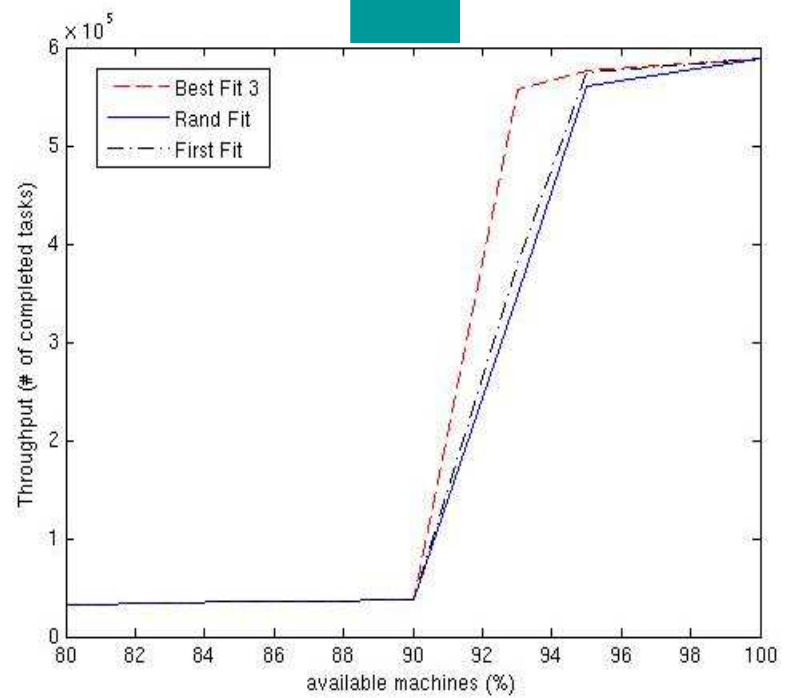
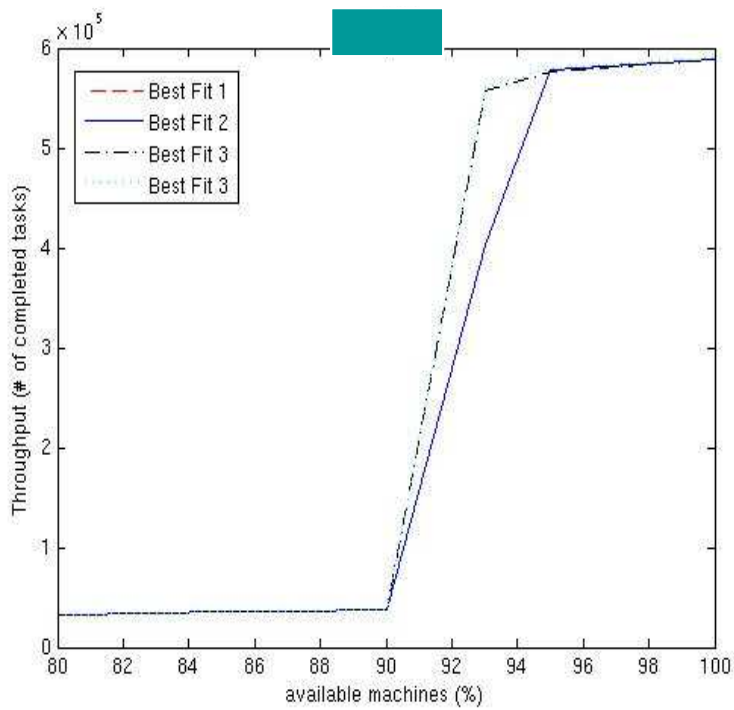
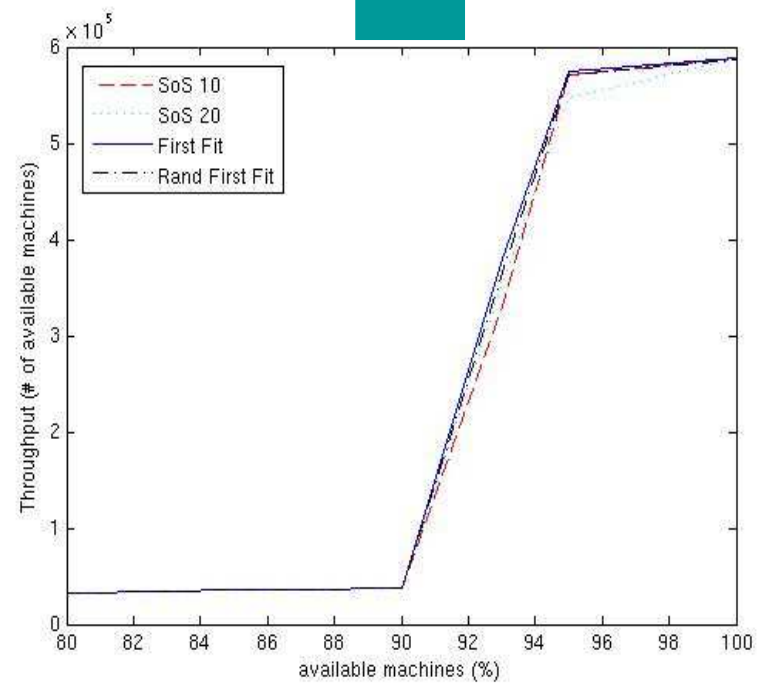
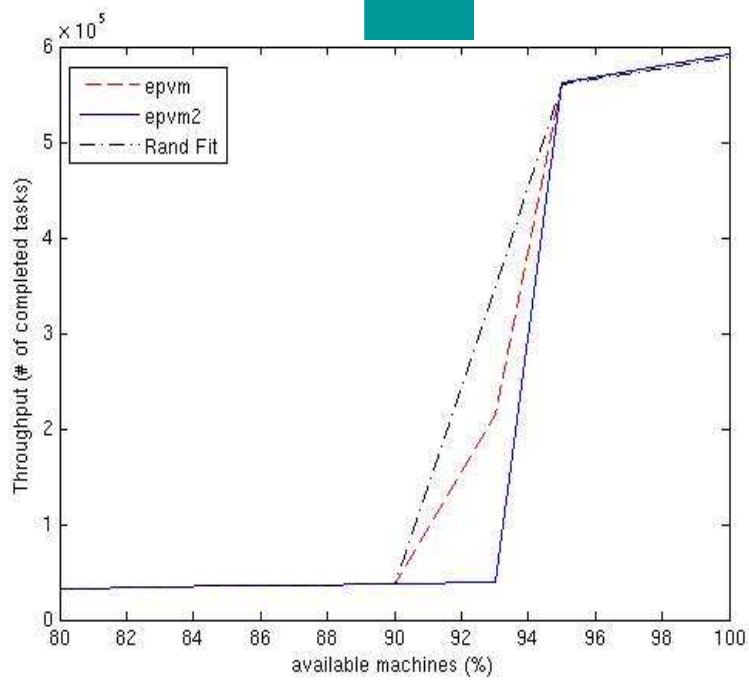
- Let $T(x)$ be the number of jobs completed using only $x\%$ of the machines in a datacenter (choose a random $x\%$).
- We can evaluate an algorithm on a cluster by looking at a collection of $T(x)$ values.
- We use 20%, 40%, 60%, 80%, 83%, 85%, 87%, 90%, 93%, 95%, 100% for x .
- Same reasoning applies to other metrics.

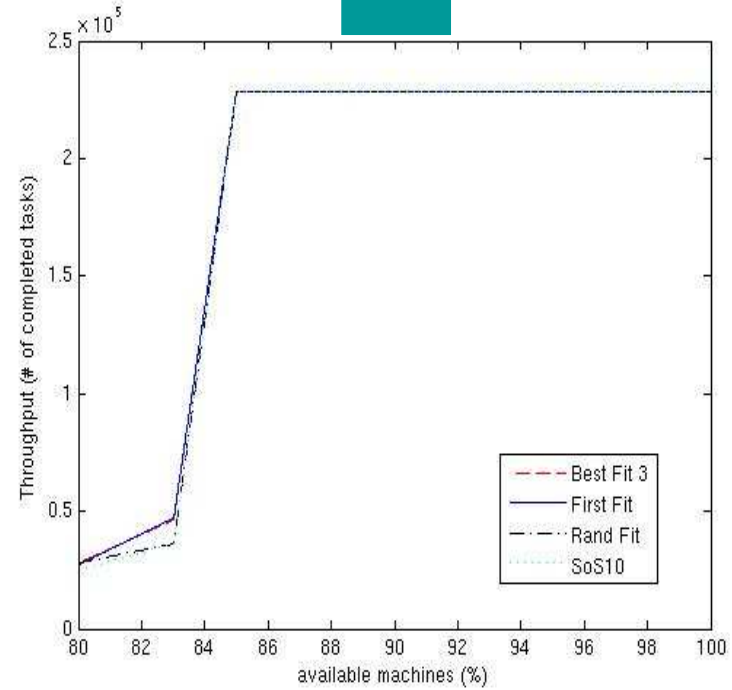
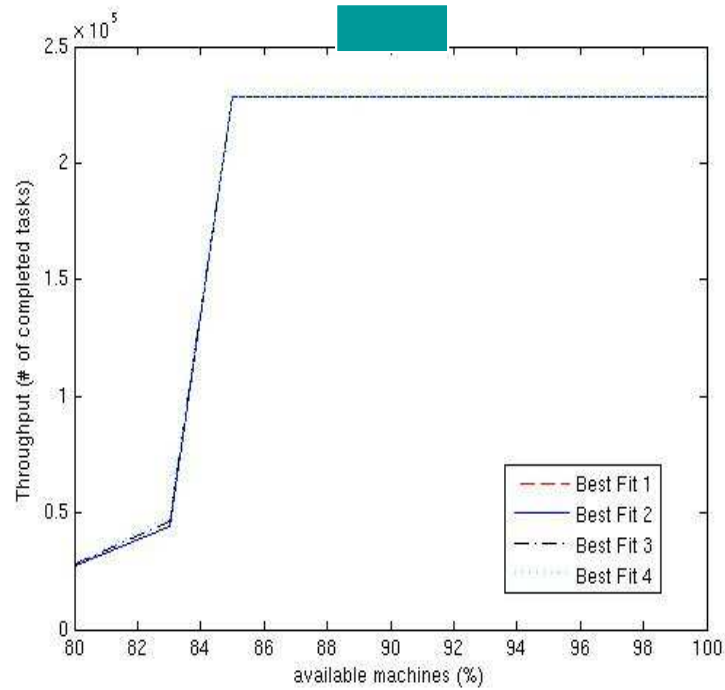
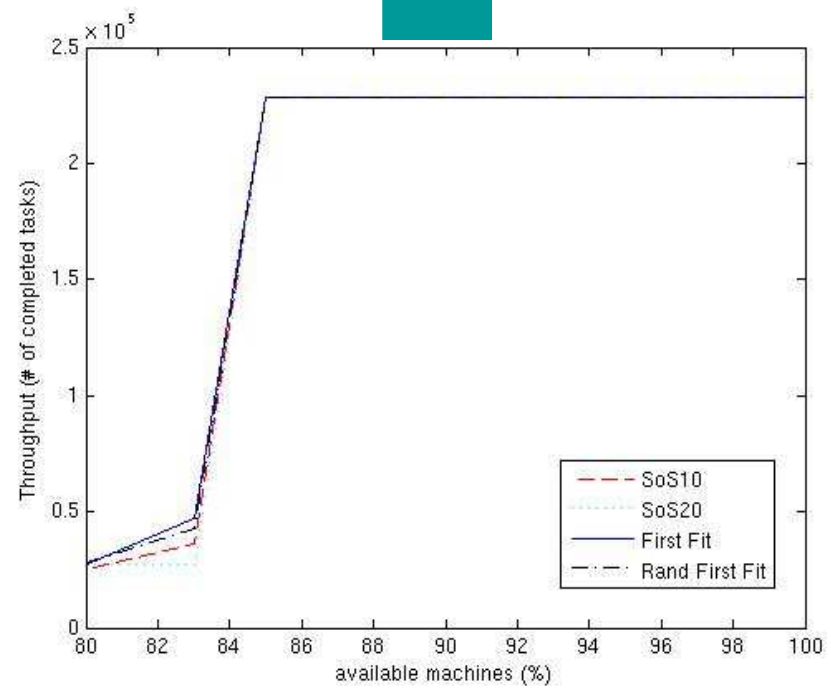
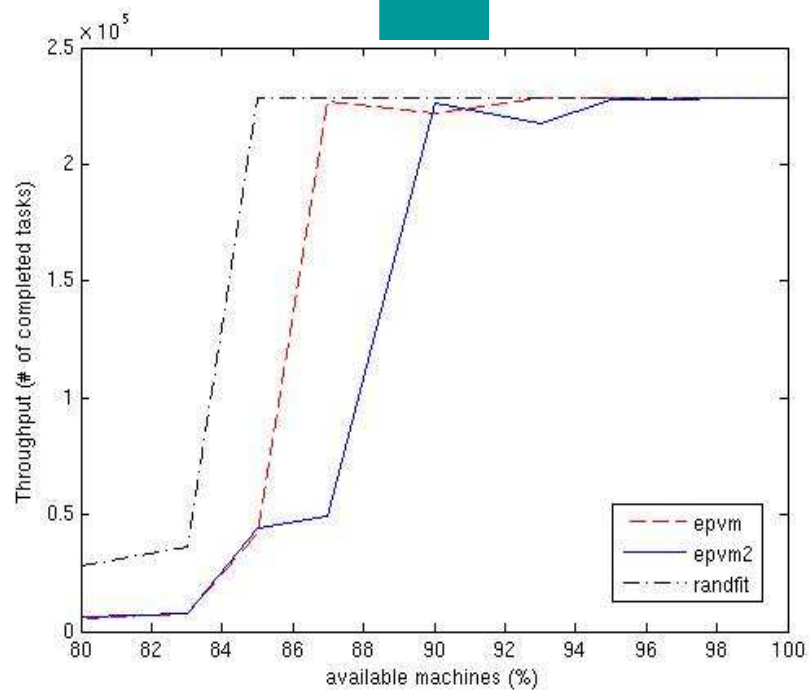












Comparison based on Throughput (multiple days on multiple datacenters)

- Over all cells and machine percentages:

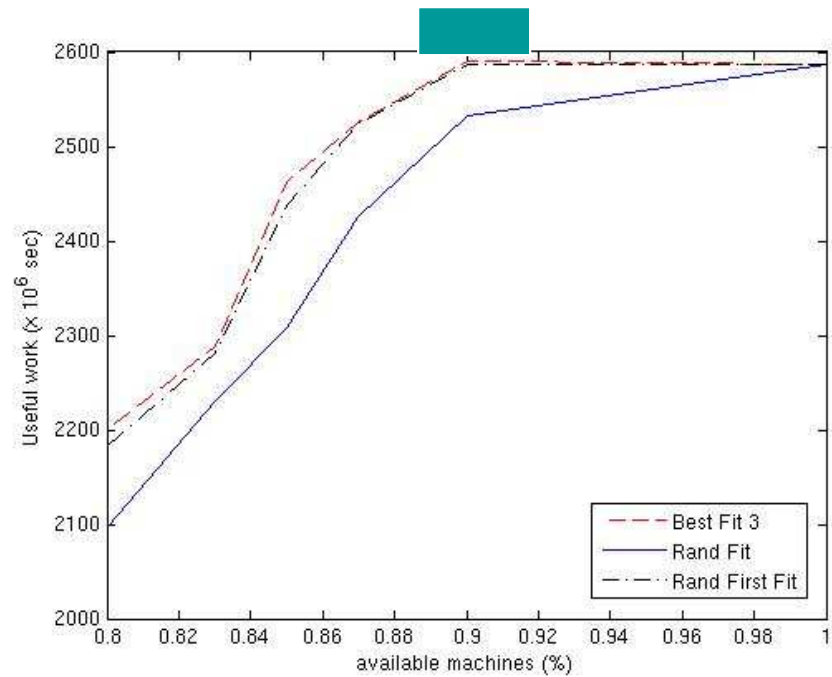
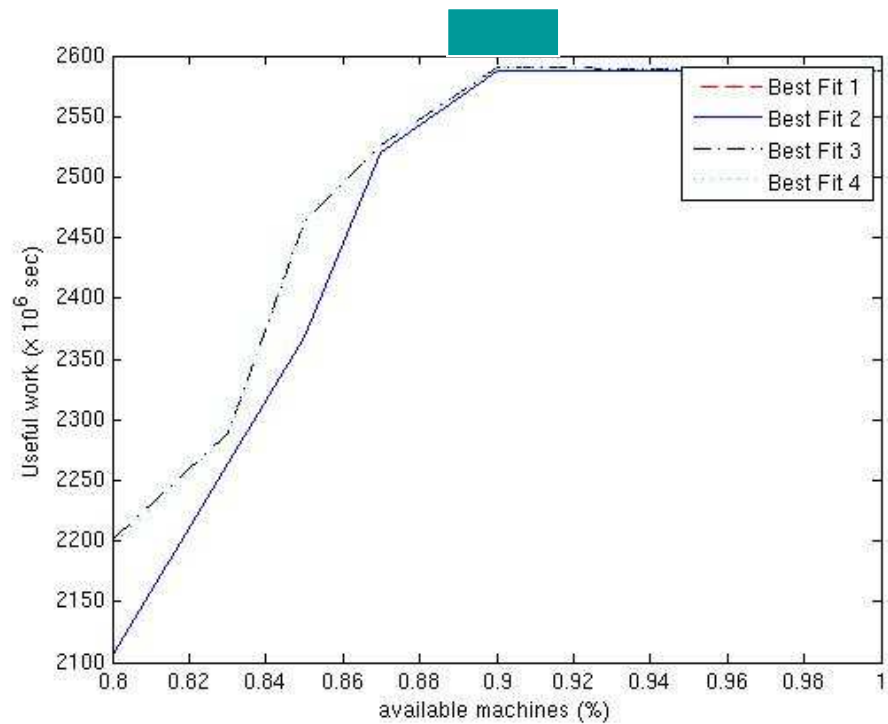
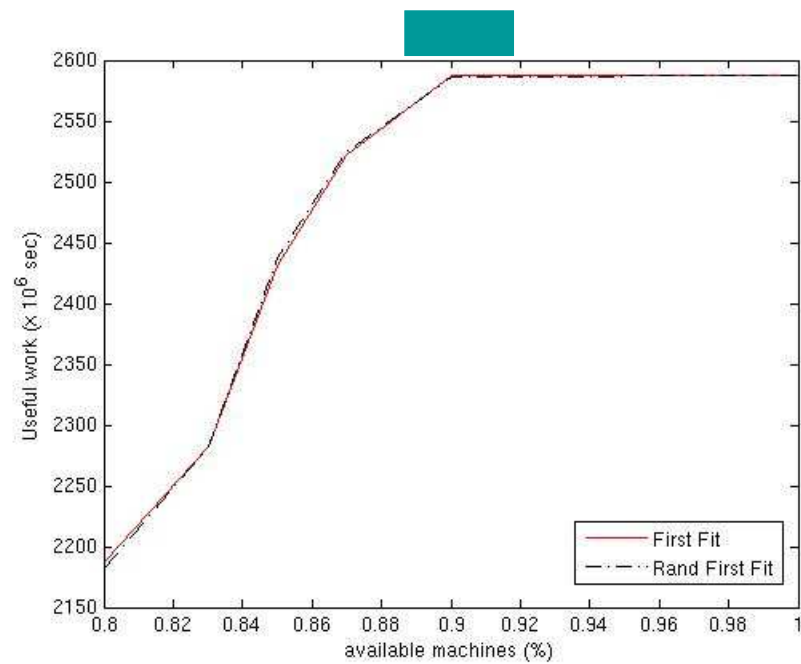
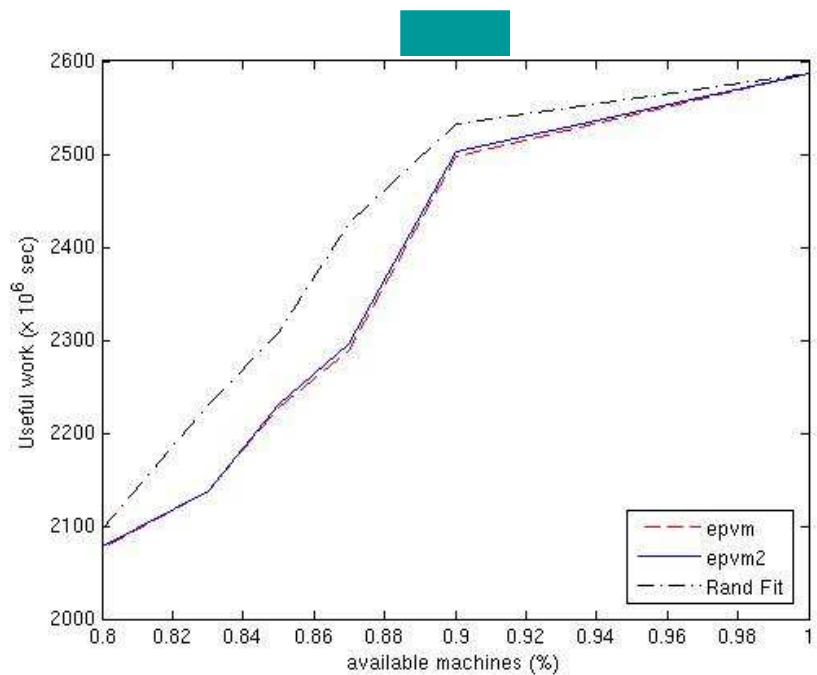
Alg	times best	times \geq 99% best
randFirstFit	31	37
SOS10	20	41
FirstFit	15	32
BestFit3	12	38
BestFit4	10	37
EPVM2	6	19
EPVM	5	35
BestFit1	5	29
BestFit2	5	29
SOS20	5	26
RandFit	5	26

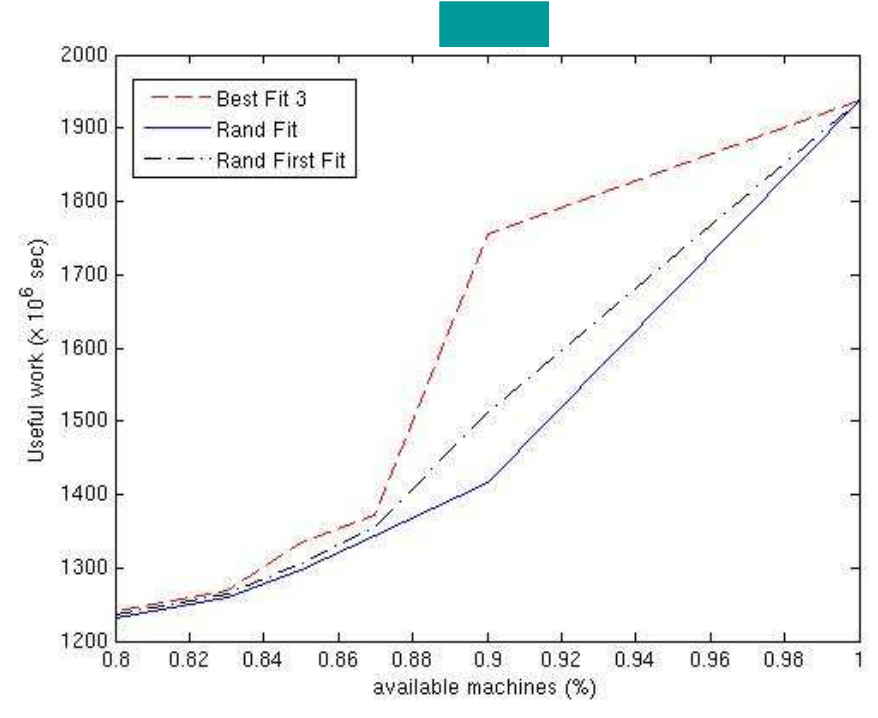
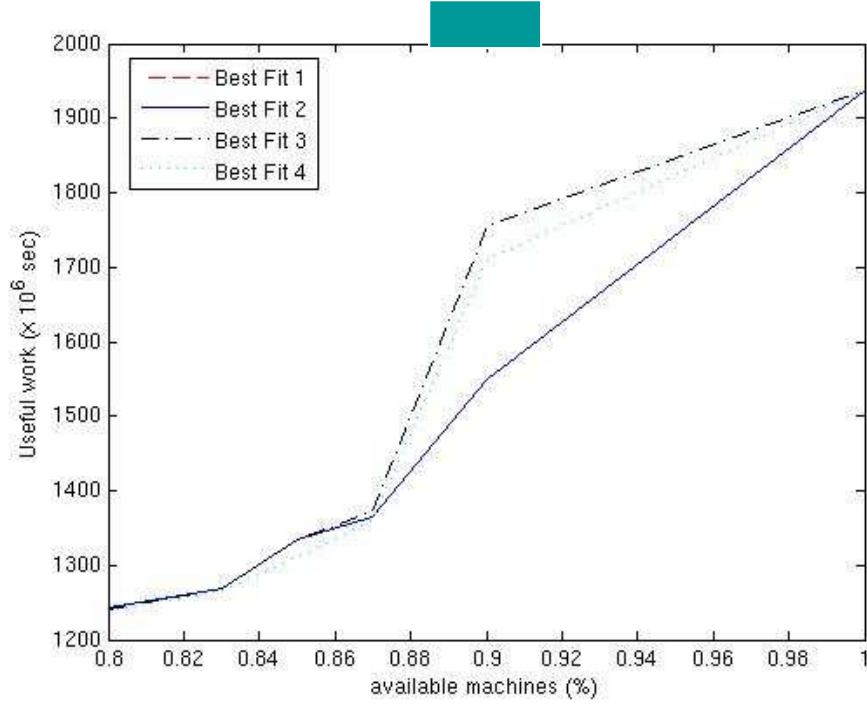
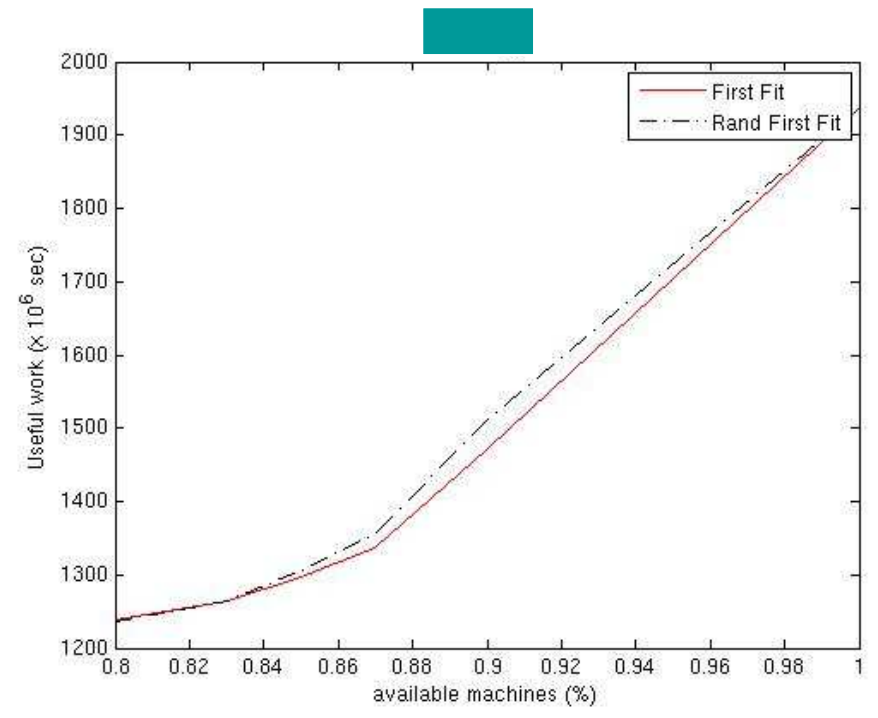
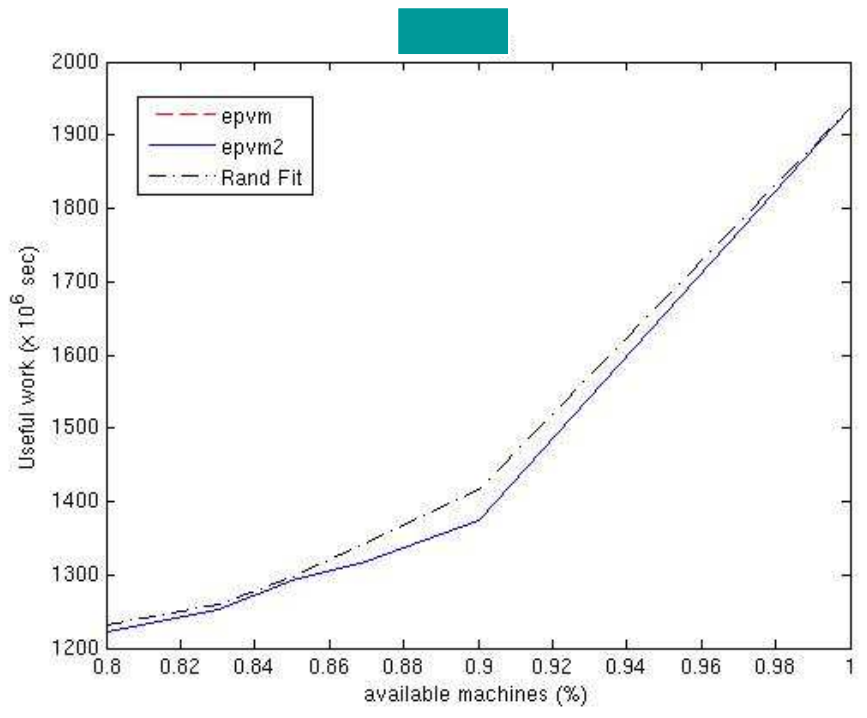
- Over all cells at 80%-90% of machines:

Alg	times best	times \geq 99% best
randFirstFit	11	16
BestFit3	10	20
FirstFit	7	15
BestFit4	6	19
SOS10	5	14
BestFit1	3	12
BestFit2	3	12
RandFit	3	12
EPVM	2	10
EPVM2	2	7
SOS20	2	12

Useful work done (in seconds)







Comparison based on Useful Work


Over all days, cells and machine percentages:

Over all days, cells at 80%-90% of machines:

Alg	times best	times \geq 99% best
BestFit3	294	318
RandFF	264	306
BestFit4	258	312
BestFit1	246	288
BestFit2	246	288
EPVM	240	270
EPVM2	240	270
RandFit	240	282

Alg	times best	times \geq 99% best
BestFit3	114	138
RandFF	84	126
BestFit4	78	132
BestFit1	66	108
BestFit2	66	108
EPVM	60	90
EPVM2	60	90
RandFit	60	90

Simulation Conclusions

- Many more experiments with similar conclusions.
 - Bestfit seemed to be best.
 - Sum-of-squares was also competitive.
 - First Fit was a little worse than sum-of-squares.
 - Worst-Fit seemed to do quite poorly.
- 

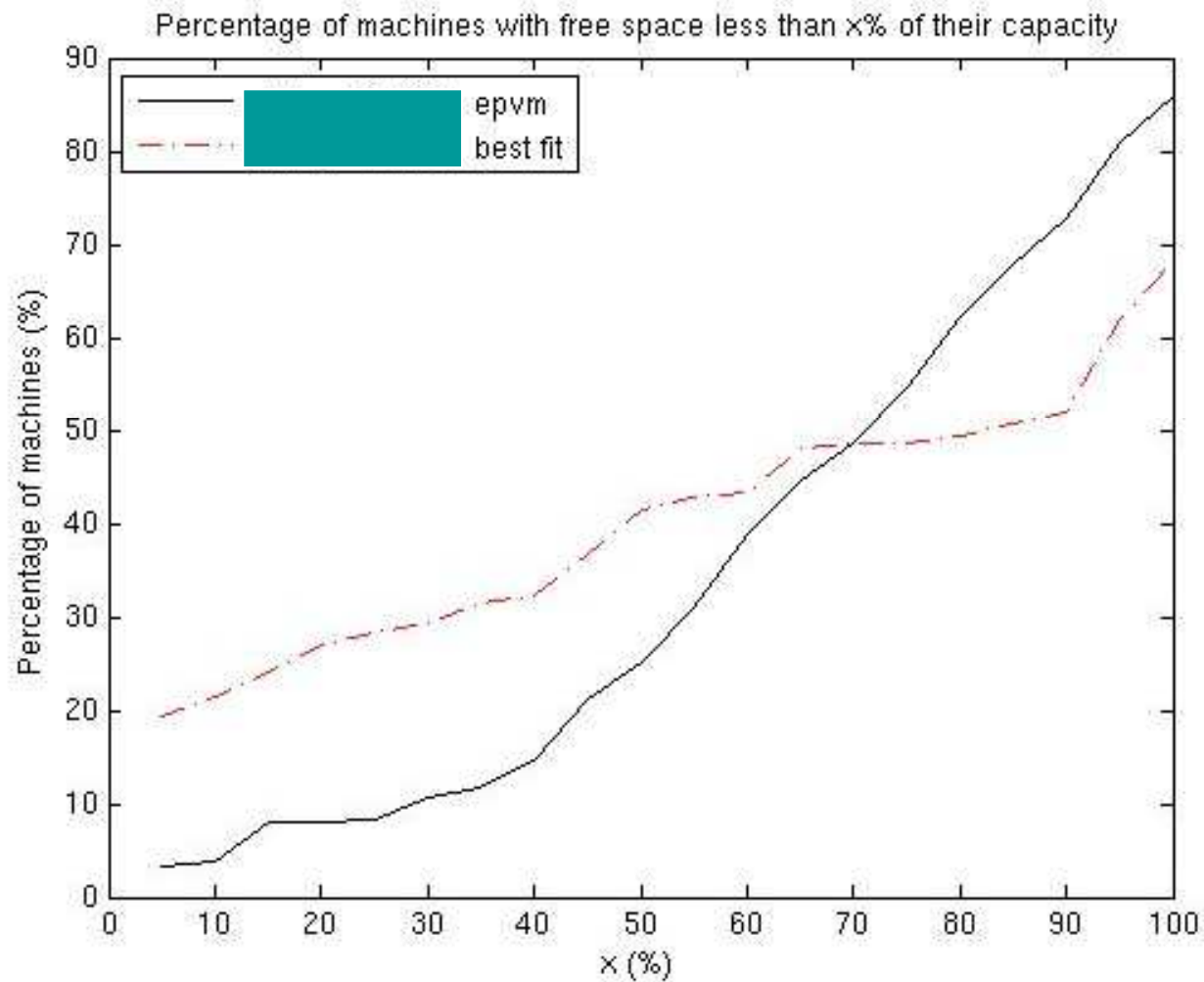
Machine Fragmentation

Thesis:

- Empty machines are good.
- Machines with large holes are good.
- Machine "fullness" can be drastic depending on the algorithm used.

- We count machines m for which
$$\text{free_cpu}(m) < (x/100) * \text{total_cpu}(m)$$
$$\&\& \text{free_ram}(m) < (x/100) * \text{total_ram}(m)$$

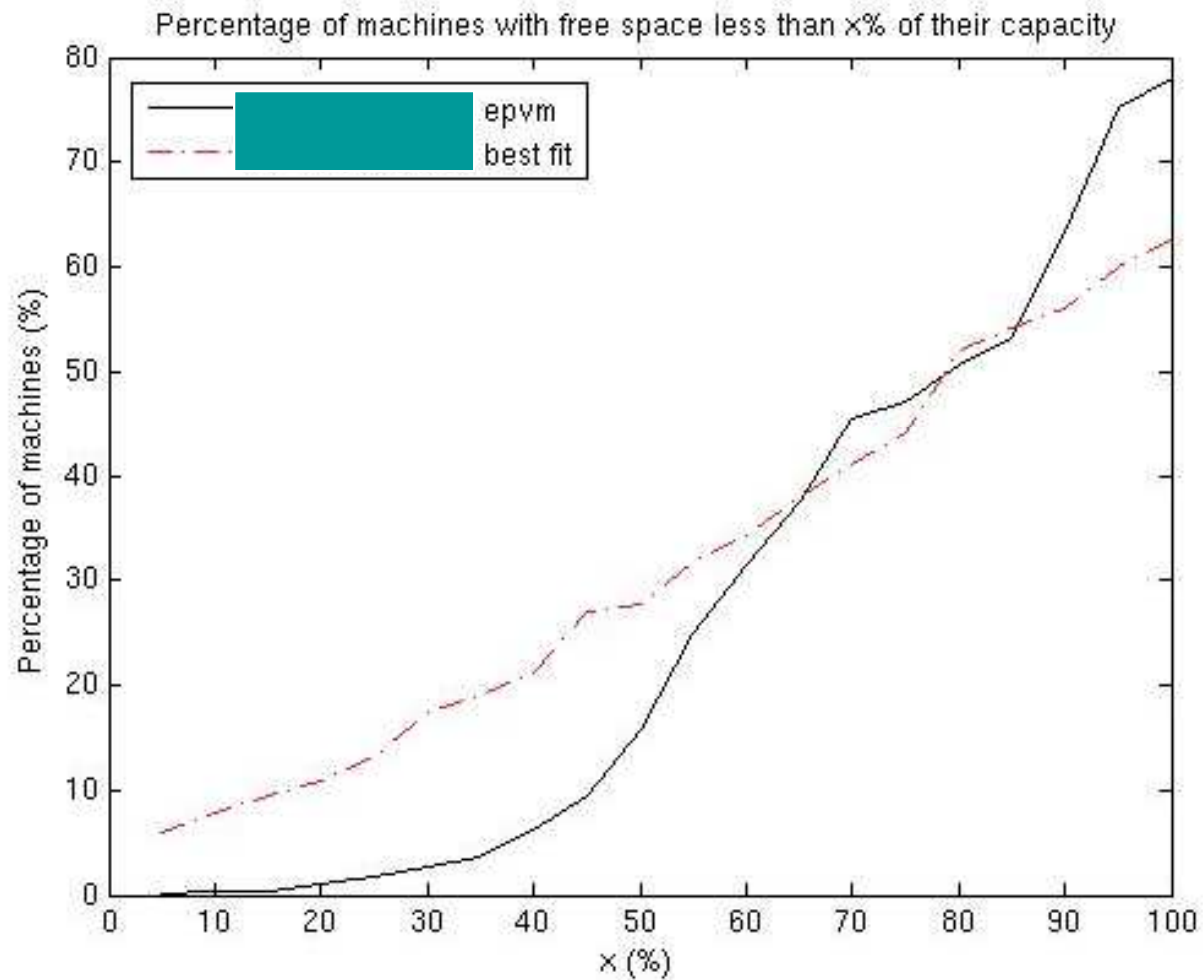
Machine Fragmentation



full

empty

Machine Fragmentation



full

empty

Power

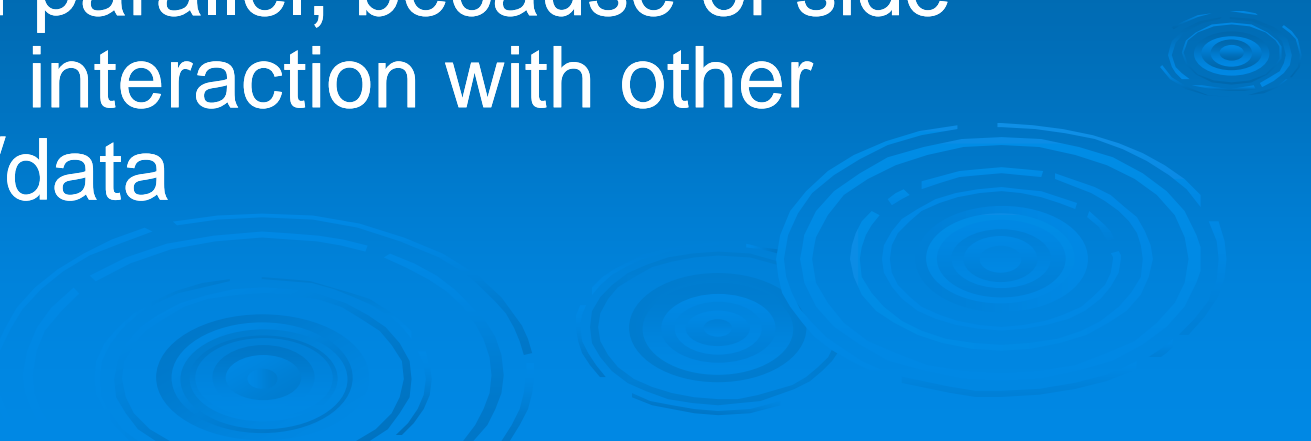
- Machines have the following power characteristics:
 - Between 50% and 100% utilization, power use is linear in machine load
 - At 0% you can turn the machine off
 - In between 0% and 50%, power usage is inefficient
- By looking at the fragmentation, you can analyze power utilization

How to do real experiments

- Simulator is only working on a model, would like live experiments.
- Ideal experiment: Run two datacenters on the same real data and compare performance.



Live Experiments on same data are difficult

- Running two real sized datacenters on same data is expensive, or even impossible
 - Once you run on small datacenters, you are forced to model your input
 - It is not clear how to run on real data on two sites in parallel, because of side-effects and interaction with other computing/data
- 

Another idea

- Look for 2 datacenters on two days that have “similar data.” Run different algorithms and compare.
- Hard to find such datacenters.



Conclusions and Future Directions

- Careful study and experimentation can lead to more efficient use of a large datacenter.
- Best Fit seems to be the best performer for the environments we studied. (Usually the best, never far from best.)
- SOS and first fit are also reasonable choices.
- Methodology for real-time testing of scheduling algorithms is an interesting area of study.
- Algorithms based on our work are now running at google and have led to improved throughput and energy performance