Seminar in Distributed Computing

Task assignment with unknown duration

Thibaut Britz - britzt@student.ethz.ch
Overview

- Model
- Performance goals
- Common task assignment policies
- Heavy tails
- Pareto Distribution
- Bounded Pareto Distribution
- TAGS Algorithm
- Results for the case of 2 hosts
- Results for more than 2 hosts
- Effect of the range of task sizes
- Server expansion metric
- Conclusion
Model

Model:

- Distributed server system with identical machines
- No cost for dispatching jobs
- Jobs not preemptible
- Service demand not known
Performance goals

Goals to achieve:

Primary:
- Minimize mean response time.
- Minimize mean slowdown
  \[ \text{slowdown} = \frac{\text{waiting time}}{\text{service requirement}} \]

Secondary:
- Guarantee fairness
  All jobs should experience the same expected slowdown.

(Note: Minimizing the total running time of all the jobs is not a goal)
Common task assignment policies

- **Random:**
  h hosts, each job gets assigned to a host with probability $1/h$.
- **Round robin:**
  $i$th job assigned to host $i \mod h$.
- **Shortest-queue:**
  job immediately dispatched to the host with the shortest queue.
Common task assignment policies

- **Least-work-remaining:**
  Send the job to the host with the least remaining work. 
  -> However, we don't know the job size.

- **Central-queue:**
  Keep one global queue and dispatch job to the next free host. 
  This policy is equal to the least-work-remaining policy and the optimal solution for exponential job size distribution.
Measurements indicate that the job distribution is not exponential, but heavy-tailed.

Heavy tailed distribution: \( \Pr\{X > x\} \sim x^{-\alpha} \)
Pareto distribution

• **Pareto** probability mass function approximates heavy-tail property:

\[ f(x) = \alpha x^{-\alpha-1} \quad x \geq 1, \ 0 \leq \alpha \leq 2 \]

• The lower \( \alpha \), the more variable the distribution

3 Properties:
• Decreasing failure rate: The longer a job has run, the longer it is expected to continue running.
• Infinite or near infinite variance
• **Heavy tail-property**: One tiny fraction of the very largest jobs comprise more than half of the total load.
Bounded Pareto distribution

- Empirical results show that job size distributions often have $\alpha \approx 1$.
- Upper bound on process size
- We can approximate the distribution of job sizes with the **Bounded Pareto** distribution probability density function $B(k, p, \alpha)$:

$$f(x) = \frac{\alpha k^\alpha}{1 - (k/p)^\alpha} x^{-\alpha - 1} \quad k \leq x \leq p, \quad 0 \leq \alpha \leq 2$$

- $k$: shortest possible job
- $p$: longest possible job
- $\alpha$: variance parameter
Bounded Pareto distribution (ctd.)

- Heavy-tail property and decreasing failure rate still valid.
- We will vary $\alpha$ from 0 to 2. $E(X)$ will be fixed to 3000 and $p$ to $10^{10}$.

- If workload heavy-tailed, the second moment “explodes”.

![Graph of Second Moment of Bounded Pareto Distribution](image)
h is the number of hosts (numbered 1..h). The $i$th host has a number $s_i$ associated with it, where $s_1 < s_2 < ... < s_h$

All jobs are immediately dispatched to Host 1 where they are serviced in FCFS order. If the job hasn't finished after $s_1$ amount of time, it is canceled, and queued at host 2, where it is restarted from scratch.
The TAGS algorithm (ctd.)

3 flavors:
• TAGS-optimize-slowdown
• TAGS-optimize-waitingtime
• TAGS-optimize-fairness

Each one of those has a different cutoff times $s_i$. $s_i$ depends on the parameters $\alpha$, $k$, $p$ and $\lambda$ (the arrival rate) and optimize the mean slowdown, waiting time, or fairness.
Analytic results for the case of 2 hosts

System load = Outside arrival rate * Mean job size / number of hosts
System load fixed to 0.5
Analytic results for the case of 2 hosts (ctd.)

Why does it perform so well?
- **Variance reduction**
- **Load unbalancing** instead of **load balancing**
Variance reduction

Variance reduction reduces the variances of job sizes that share the same queue. This improves performance since it reduces the chance of a small job getting stuck behind a long job in the same queue.

- **Property:** For a single FCFS queue, mean queue waiting time, slowdown and queue length are all proportional to $E(X^2)$.
- **Random task assignment:**
  Performance metrics stay proportional to $E(X^2)$ of $B(k,p,\alpha)$. Since $E(X^2)$ is high, performance is poor.
- **Least-work-remaining (central-queue):**
  Mean queue length, and therefore mean waiting time and mean slowdown proportional to $E(X^2)$.
- **TAGS:**
  Reduces the variance of job sizes at the individual hosts. Since the service time of host $i$ is capped at $s_i$, $E(X^2)$ of each host $i$ is lower than $E(X^2)$ of the original $B(k,p,\alpha)$ distribution. (Except for the last host)
Load unbalancing

- TAGS tries to unbalance load.
- All other policies try to balance load.

Observations:
- $\alpha < 1$: host 1 is underloaded
- $\alpha \approx 1$: Load is balanced
- $\alpha > 1$: host 2 is underloaded
Load unbalancing (ctd.)

Why is load balancing favorable for the mean system slow down?
- Heavy-tail property.

- $\alpha < 1$: Very small fraction of jobs is needed to make up half the load. Because of the heavy-tail property, the load at Host 2 will be extremely high. Since most jobs run at host 1, the mean slowdown is very low.

- $\alpha \approx 1$: Distribution not as heavy tailed. Again we would like to underload host 1. A larger fraction of jobs must have host 2 as destination to create high load at host 2. But jobs at host 2 will impact more on the mean slowdown. This implies higher load at host 1 to reduce slowdown.

- $\alpha > 1$: No matter how we choose the cutoff $s_1$, a significant number of jobs will still have host 2 as their destination. So we need to keep performance of host 2 in check.
Load unbalancing (ctd.)

- How does load unbalancing optimize fairness?
- Under TAGS-optimize-fairness, the mean slowdown experienced by short jobs is equal to the mean slowdown experienced by long jobs.
- One might think of unfairness on 2 counts:
  - short jobs run on host 1 which has very low load (for low $\alpha$) and very low $E(X^2)$
  - short jobs don't have to be restarted from scratch and wait on a second line

However short jobs are short. They don't need much time to complete. Since we have a heavy-tailed distribution, longer jobs are really longer ("elephants") and can afford the longer wait.
Different loads

- Still distributed server with 2 hosts, but load varies.
Different loads (ctd.)

Observation:
• performance of TAGS correlates with load

2 Reasons:
• The higher the load, the less TAGS can unbalance the jobs. For lower $\alpha$'s, TAGS can't pile as much work at host 2 and underload host 1, since the load at host 2 must not exceed 1.
• Excess = Extra work created by restarting jobs from scratch. The excess is the difference between the sum of the loads on the hosts and $h \times$ system load.
Analytical results for more than 2 hosts

Observation:
• For 2 hosts, TAGS-optimize-slowdown was good if system load was 0.5 or less.

Claim:
• h host system with a system load $\rho$ can always be configured to produce performance which is at least as good as the best performance of a 2-host system with system load $\rho$. 
Analytical results for more than 2 hosts (ctd.)

Observations:
- Performance of random stays the same
- Performance of Least-work-remaining improved a little
- Huge improvement in performance for TAGS. Greater flexibility for choosing cutoff points.
Server expansion performance metric

- No one would run a system with slowdown of $10^5$.
- **Server expansion metric:**
  How many new hosts do we have to add to bring the mean slowdown to a reasonable level (arbitrary set to $< 3$).

(We start with a 2 host system and system load 0.7)
(example: $\alpha = 0.6$, 2 hosts $\rightarrow$ TAGS: $10^9$ ; 4 hosts $\rightarrow$ TAGS: 2, LWR: $10^8$ ; 13 hosts $\rightarrow$ LWR $< 3$)
Server expansion performance metric (ctd.)

Observations:
• For TAGS, the server expansion requirement is at most 3.
• For Least-work-remaining the server expansion ranges from 1 to 27. Still somehow good since performance increases when hosts are added.
• Random is exponentially worse than the others.
Effect of the range of task sizes

- Previous assumption was to set upper bound to $p=10^{10}$.
- What if we lower this bound to $p=10^7$. 

![Second Moment of Bounded Pareto Distribution](image1)

$p=10^{10}$

![Second Moment of Bounded Pareto Distribution](image2)

$p=10^7$
Effect of the range of task sizes (ctd.)

- Lower variance might suggest that TAGS improvement won't be so dramatic over the other assignment policies where \( p \) was set to \( p=10^{10} \)
- But still good performance.
Conclusion

• Interesting algorithm that challenges natural intuitions (eg load balancing, killing jobs).
• TAGS is outperforming other policies by several orders of magnitude if the system load is not too high.
• Normally fairness and performance conflicting goals, here they are quiet close.
• TAGS outperforms all other policies with respect to the server expansion metric.
• Raises interesting questions in out of scope fields:
  - Scheduling jobs at CPUs
  - Area of network routing