Treadmill Scheduler Optimization

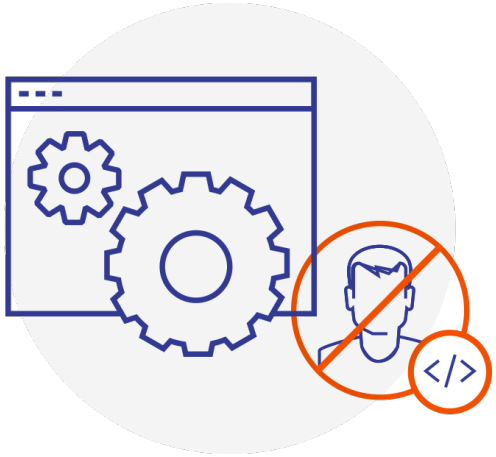
2017.08

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Outline

- Problem
- Idea
- Implementation
- Case Study
- Evaluation
- Future Work

Problem



Configurability

Extensibility

Idea

- Implement a new scheduler framework:
  - extendable
  - configurable

- Related projects or papers:
  - **Google Kubernetes**
  - Google Omega (EuroSys 13)
  - Apache Mesos (NSDI 11)
  - Apache Hadoop Yarn (SOCC 13)
  - Sparrow (SOSP 13)
  - Apollo (OSDI 14)
  - Hawk (ATC 15)
  - Mercury (ATC 15)
  - Firmament (OSDI 16)

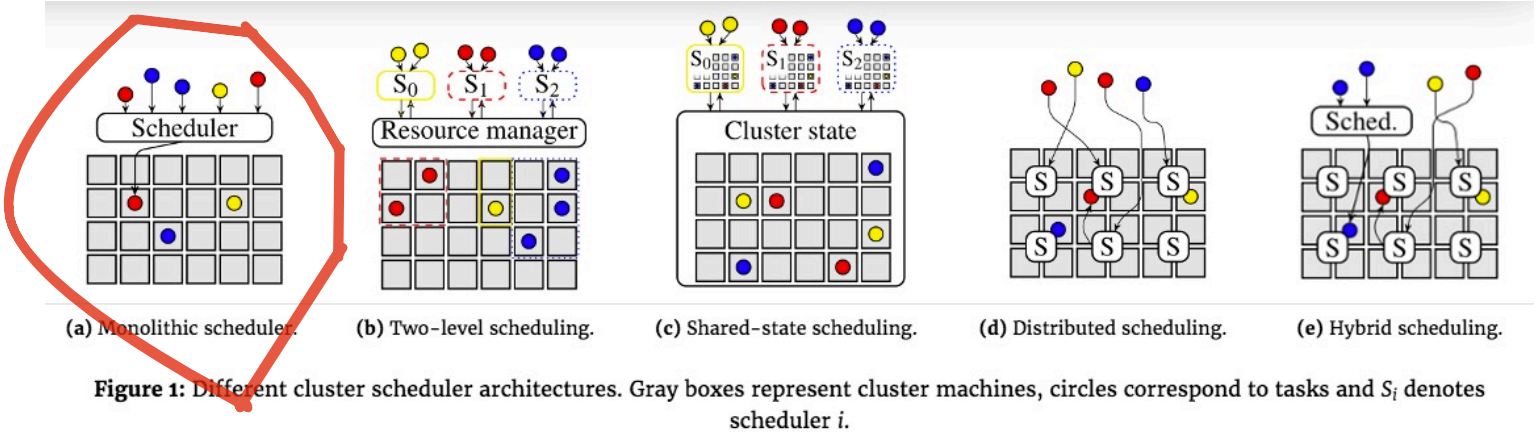
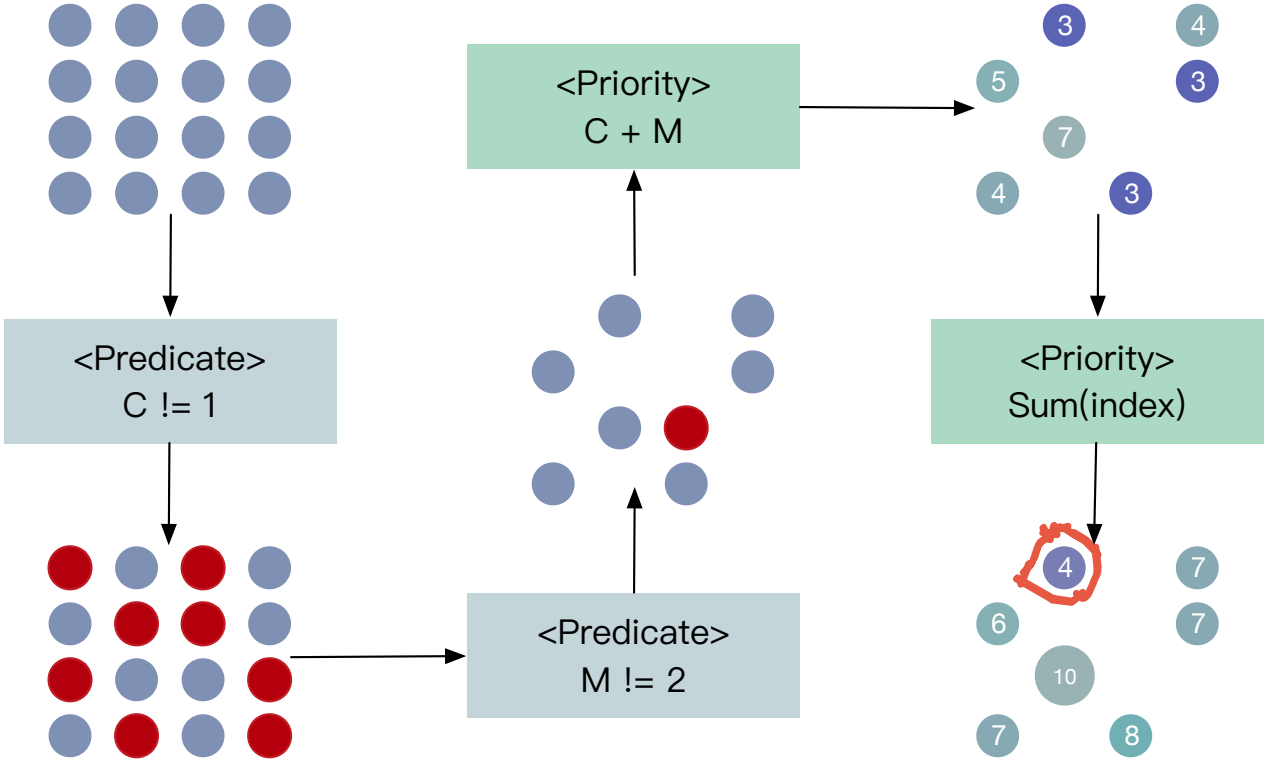


Figure 1: Different cluster scheduler architectures. Gray boxes represent cluster machines, circles correspond to tasks and S_i denotes scheduler i.

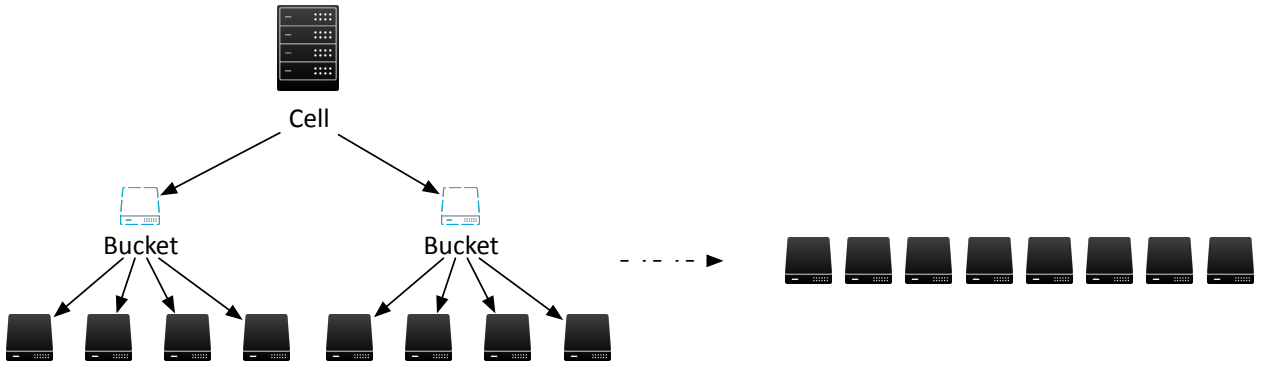
Idea


C=1
M=1
C=2
M=1
C=1
M=2
C=3
M=1
C=4
M=1
C=1
M=3
C=1
M=4
C=2
M=1
C=1
M=2
C=3
M=4
C=3
M=2
C=1
M=2
C=3
M=1
C=1
M=3
C=2
M=1
C=1
M=2

Idea

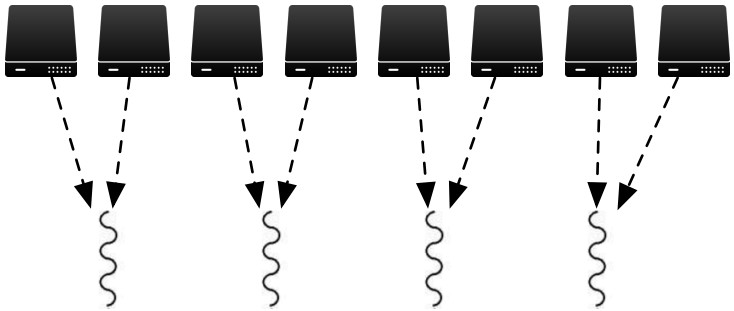

<Predicate>
C != 1
<Predicate>
M != 2
<Priority>
C + M
<Priority>
Sum(index)
3
4
5
3
7
4
3
4
7
6
7
10
7
8

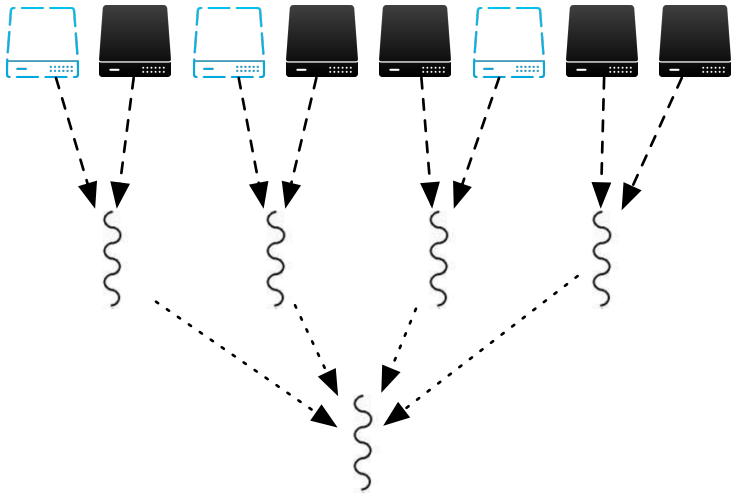
Implementation



Implementation

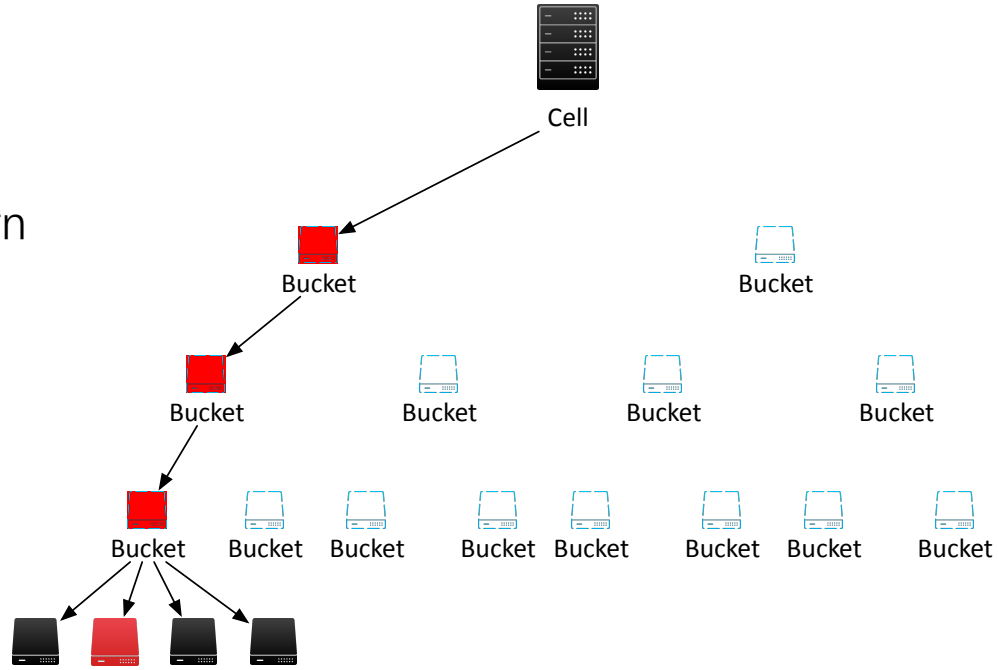
- Concurrency
  - Predicate in parallel
  - Priority in map-reduce pattern





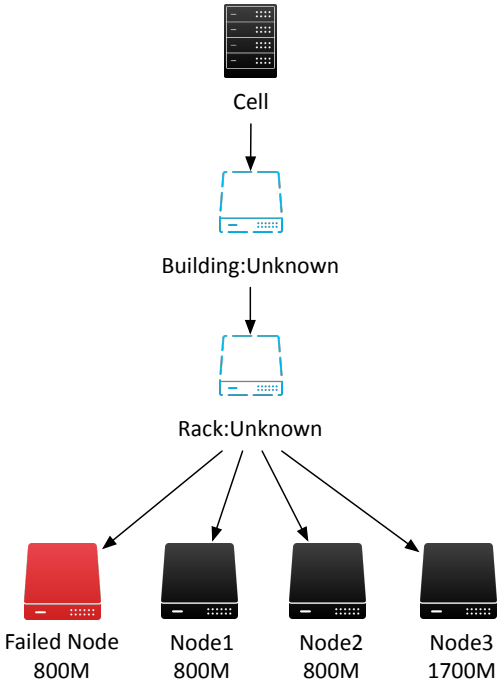
Implementation

- Concurrency
  - Predicate in parallel
  - Priority in map-reduce pattern
- No wandering tree problem



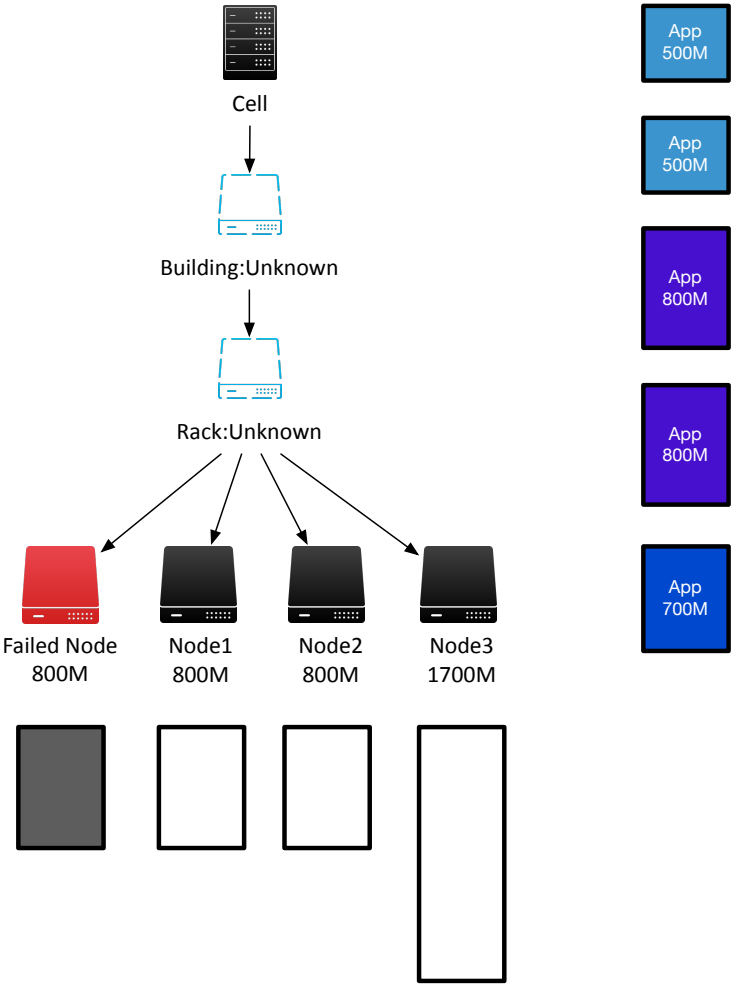
Case Study: Resource Fragmentation

- 4 Servers:
  - 3G Failed Server(800M available)
  - 3G Server(800M available) * 2
  - 4G Server(1700M available)



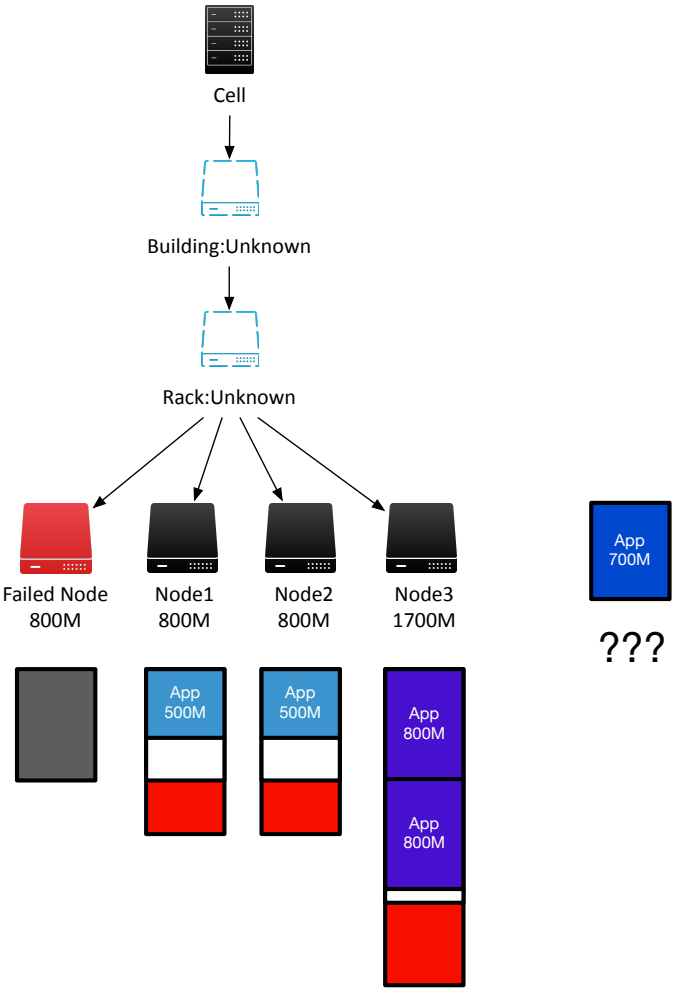
Case Study: Resource Fragmentation

- 5 Applications:
  - 500M Application * 2
  - 800M Application * 2
  - 700M Application

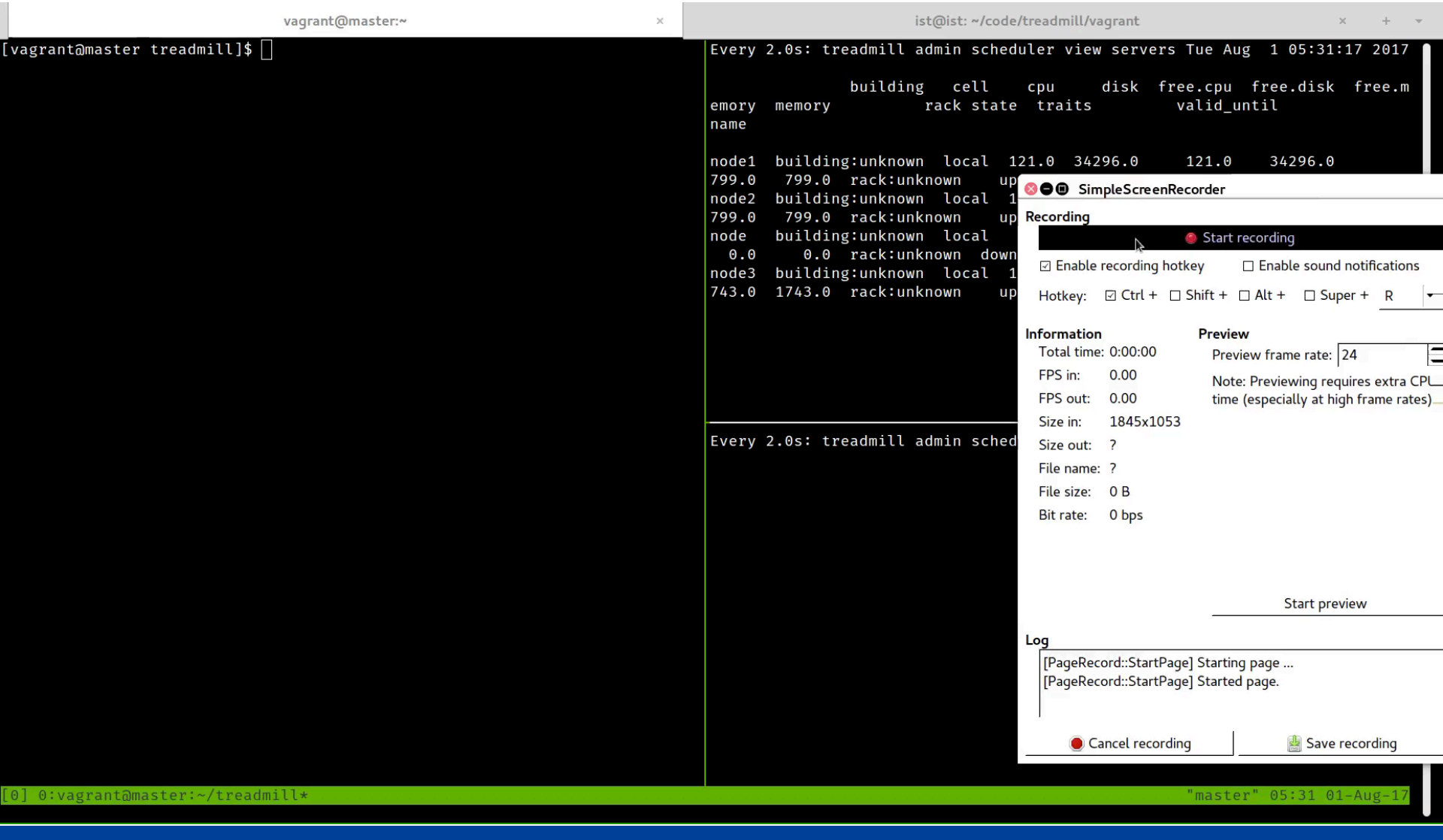


Case Study: Resource Fragmentation

- Native Scheduler: Spread

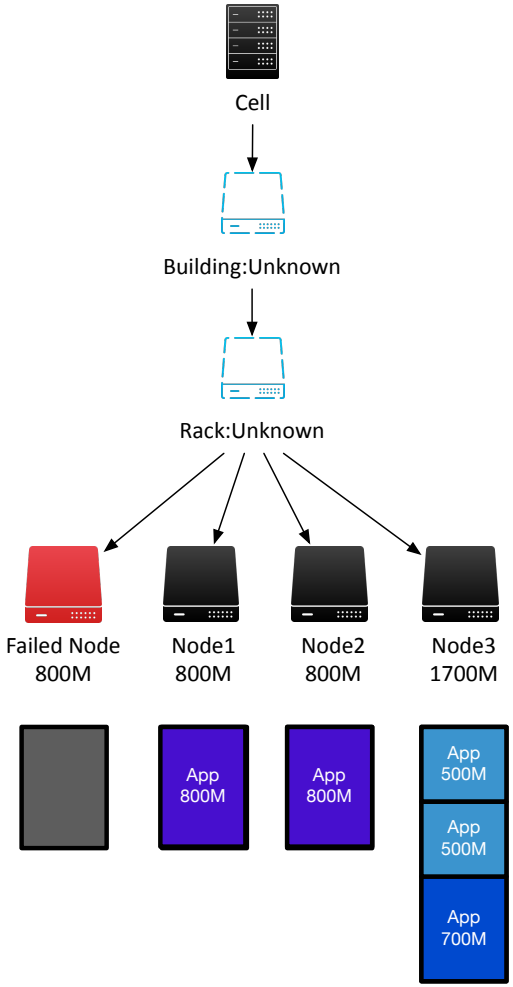


Case Study: Resource Fragmentation



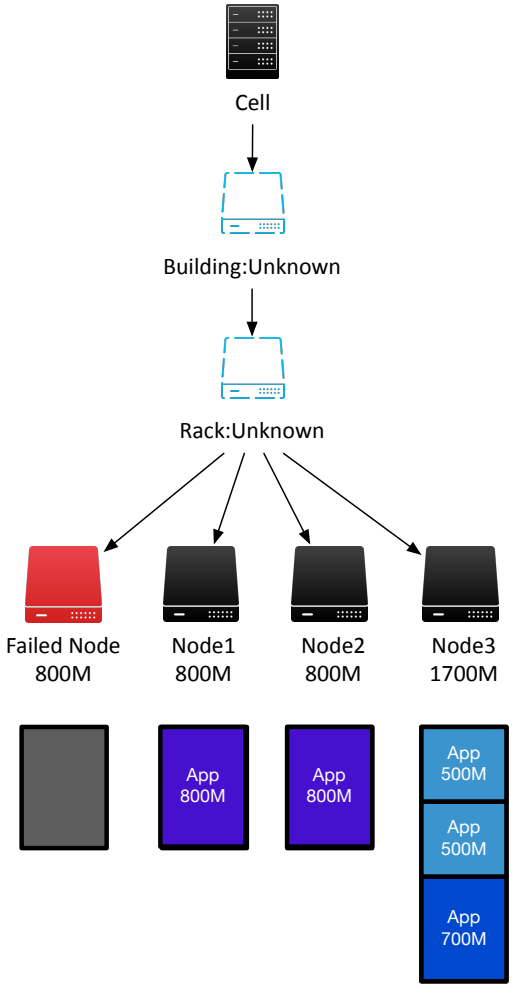
Case Study: Resource Fragmentation

- Custom Scheduler:
  - Predicates:
    - match_app_constraints
    - match_app_lifetime
    - alive_servers
    - keep_space
  - Priorities:
    - spread

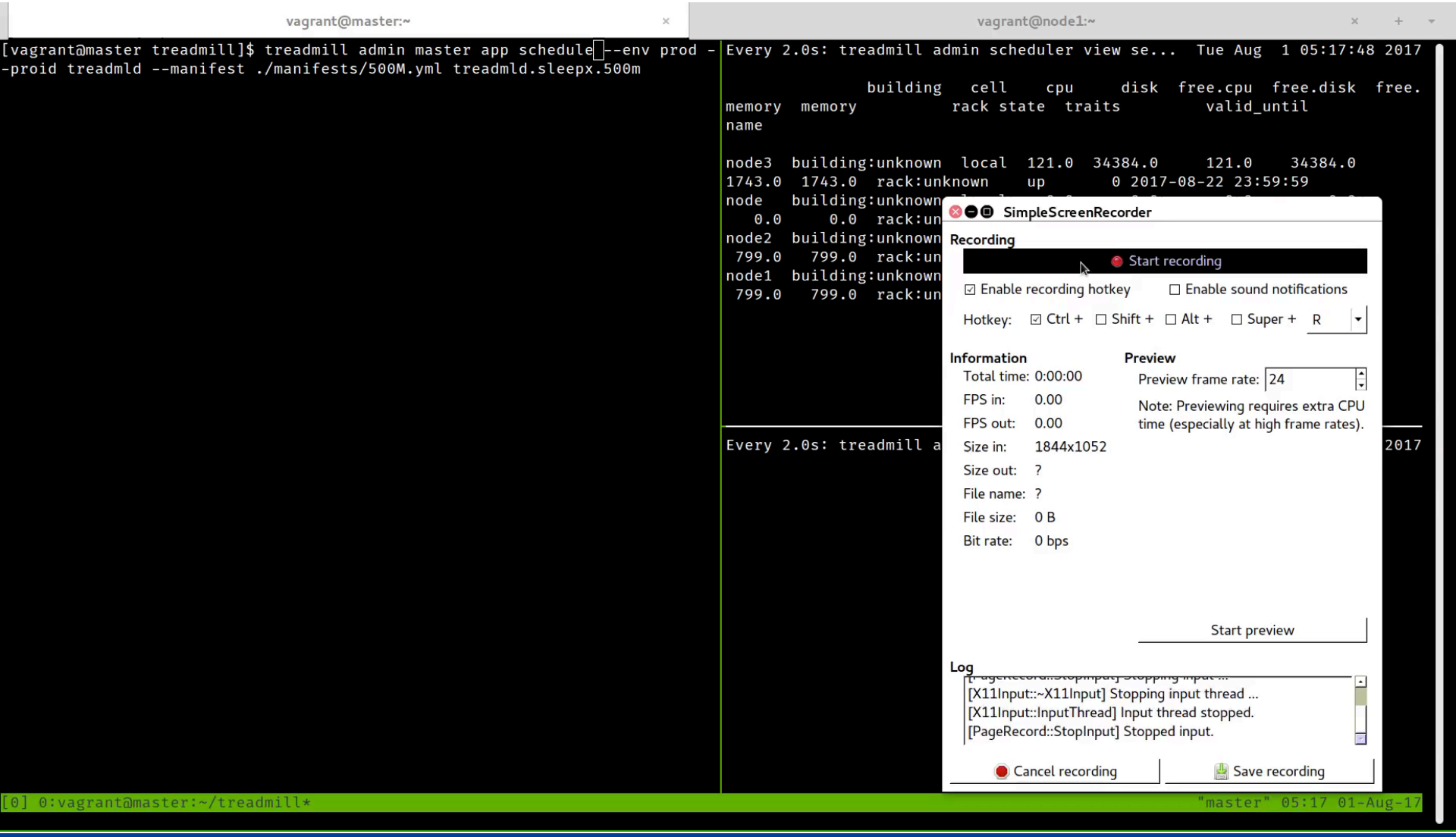


Case Study: Resource Fragmentation

- alive_servers: Filter the failed nodes
- keep_space: Reject the applications which will come with resource fragmentation

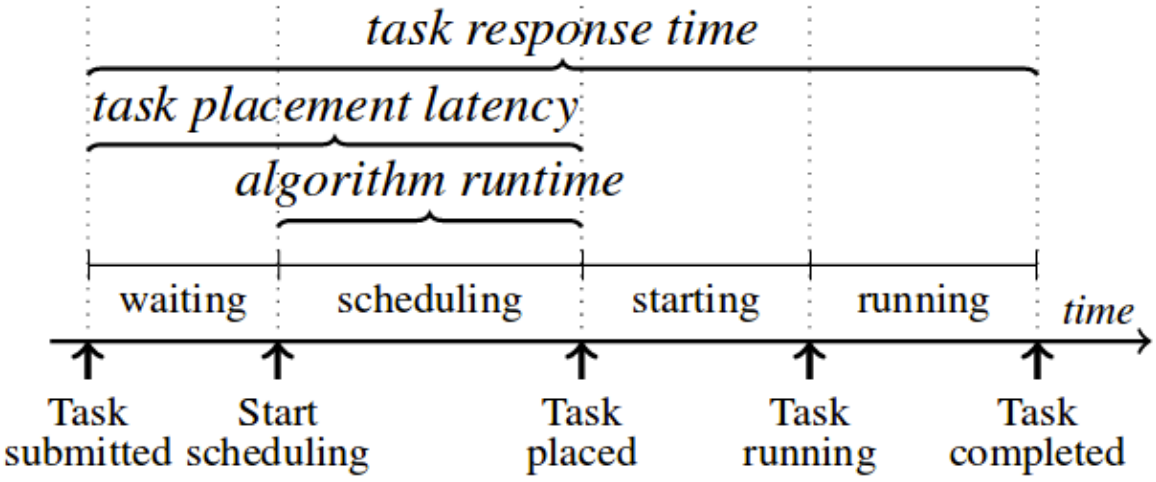


Case Study: Resource Fragmentation



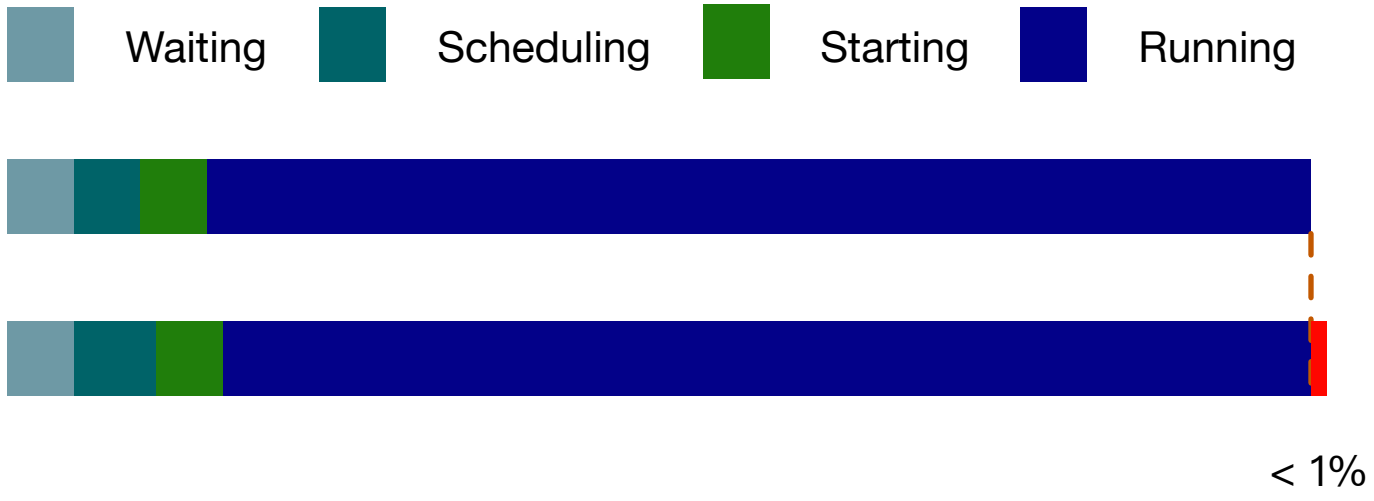
Evaluation

- It takes more time to schedule a job but it does not matter since the majority of jobs in treadmill are long-running.
- More in our technical report



Evaluation

- It takes more time to schedule a job but it does not matter since the majority of jobs in treadmill are long-running.
- More in our technical report



Progress of Development (Phase 1)

- **Start scheduler process in master, and start the necessary services in server.** It can schedule and run apps but has some problems when the app should be cleanup. (History in GitHub)
  - Read related papers and code (Before Mar 7)
  - Try to run treadmill, use ApacheDS as LDAP server, but the code in public repo could not work. (Mar 7)
  - Try to run scheduler process in master separately. Remove LDAP. (Mar 13)
  - Use `./bin/treadmill --debug admin master server configure` and `./bin/treadmill --debug admin master app schedule` to allocate server and app. (Mar 17)
  - Import vagrant, and **use bash script to start network service, cgroup service and local disk service in server side.** Init server with `./bin/treadmill sproc init`, instead. (Mar 20)

Progress of Development (Phase 1)

- Hack eventmanager (Mar 22)
- Hack app config manager(Apr 6)
- Hack supervisor (Apr 12)
- Add set up script to set up the environment in vagrant VM. And do some hacks to run server without errors (Apr 22)
- Fix some bugs in local-up bash script, and run rrdcached in server side (Apr 24)
- Add bash script to export environment variables (May 4)

Progress of Development (Phase 2)

- Try to use vagrant maintained by TW, but it does not work well. **Start to write benchmarks and simulator for treadmill scheduler, and use R Language Notebook to present the result.** (History in GitHub)
  - Implement the first benchmark. (May 19)
  - Record the result of the benchmark, and profile scheduler. Show the result in R Language Notebook. (June 1)
  - Add two benchmarks (June 5)
  - Generate technical report (June 15)
  - Fix bugs in the graph (June 22)

Progress of Development (Phase 3)

- **Implement the core logic of the new scheduler framework** (History in GitHub)
  - **Add configuration logic of the scheduler**, which is the main functionality of the scheduler framework (July 10)
  - **Add predicates and priorities support** (July 11)
  - Fix the bug about application placement in the server (July 12)
  - Hack the Cron API to allow users to call `treadmill admin invoke` (July 14)
  - Fix test_affinity_limits case and refactor the logic about flatten nodes (July 15)
  - Add CLI options to switch the type of scheduler and move code about algorithm to plugins and add CLI options to pass configurations to the new scheduler (July 16)
  - Update technical report (July 19)
  - **Fix all test cases** (July 22)
  - Separate the new test cases with the old (July 24)
  - Refactor the code about affinity limits (July 25)

Summary

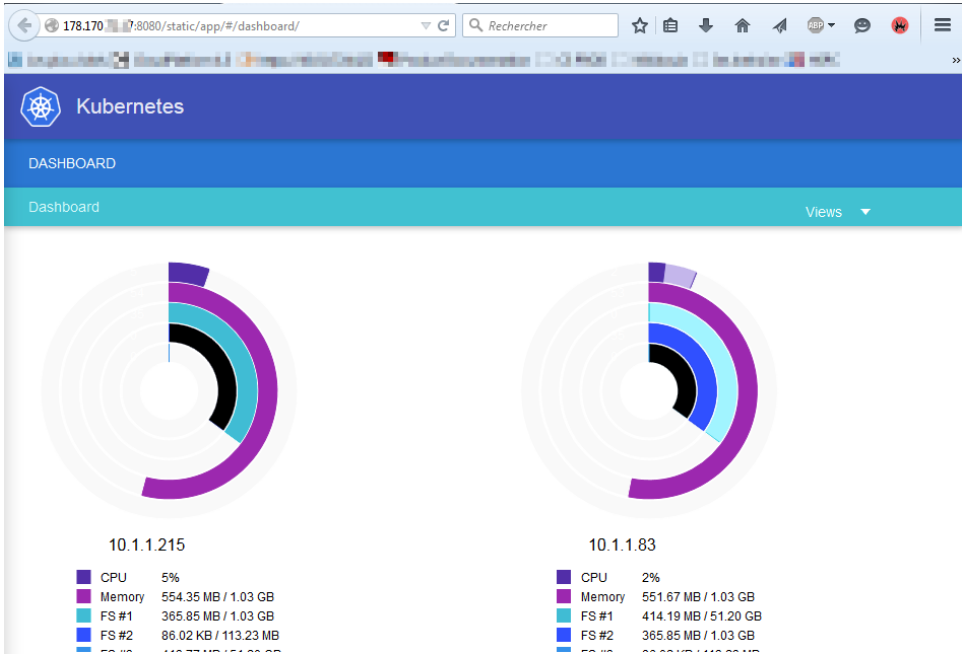
- New scheduler framework
- Limited support for scheduling algorithms because of the lack of node monitor

Future Work

- Monitor
- Customization
- Advanced Features
- Hybrid Scheduler

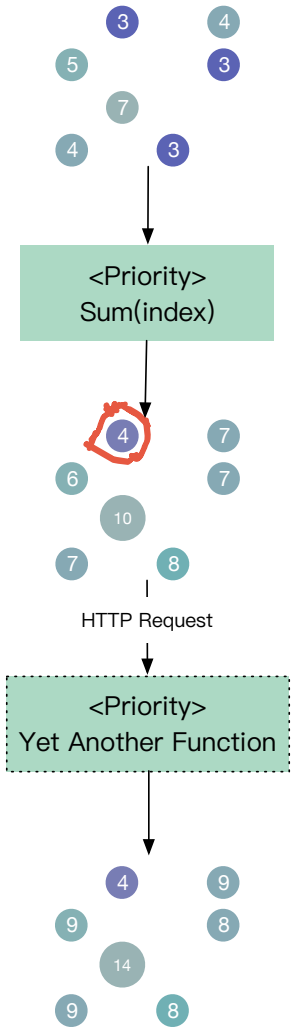
Monitor

- Basis of many features:
  - Resource Oversubscription
  - Fine-grained and Precise Scheduling
  - Application Lifecycle Management



Scheduler Customization

- Runtime Configurability
  - Keep the configuration in ZooKeeper and watch the configuration to update
- Extendable Architecture
  - Pluggable
  - But hurt the scheduler latency and throughput



Advanced Scheduler Features

- Affinity, Inter-Affinity & Anti-Affinity Scheduling
  - More expressive than label
  - More flexible
  - Affinity
    - Scheduling an application on to a node
  - Inter-affinity
    - Co-locating applications on a same node on account of some reasons, for example dependency, network latency.
  - Anti-affinity
    - Spreading applications on different nodes

Hybrid Architecture

- Centralized with distributed
  - Centralized scheduling for long-running jobs
  - Distributed scheduling for short jobs

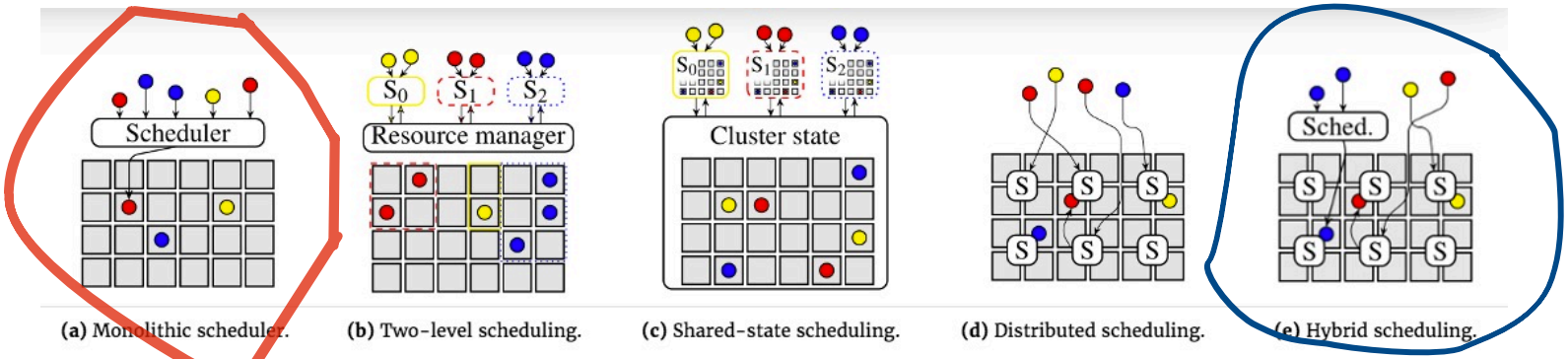


Figure 1: Different cluster scheduler architectures. Gray boxes represent cluster machines, circles correspond to tasks and S_i denotes scheduler i.

A lot of other features...

- Suboptimal scheduling
- Concurrent schedulers with different names
- Resource arbitration to solve potential conflicts
- Scheduling based on min-cost max-flow optimization
- ...

Acknowledgments

- Thank Walt and Xincheng for their guidance.

Thanks