Towards Emergent Scheduling for Distributed Execution Frameworks

Paul Allan Dean

Supervisor: Dr Barry Porter

Lancaster University, UK

p.dean1@lancaster.ac.uk

Introduction

As data collection, storage, and processing capabilities have increased, it is now common to process many gigabytes of data – for analysis, information extraction, or machine learning.

Distributed execution frameworks (DEFs) like Spark provide users with a framework for submitting, scheduling, and executing large complex workloads across hundreds of machines.

- Workloads consist of multiple jobs
- A job is a set of one or more computational tasks.

The main cost of DEFs is the *scheduling of workloads*, with a reduction in this cost offering:

- Lower completion time
- Efficient resource usage
- Reduced energy consumption

Previous Approaches

General purpose DEFs

- Apache Spark

In-memory Data intensive workloads

- Flink

Prioritises latency sensitive workloads

- Hadoop

Long running batch processes

- Single fixed architecture using a single policy
- Same scheduling for all workloads

Previous Approaches

General purpose DEFs	Workload specific DEFs
• Apache Spark In-memory Data intensive workloads • Flink Prioritises latency sensitive workloads • Hadoop Long running batch processes	• Ray Prioritises reinforcement learning workloads • Storm Low-latency/real-time task computation • Horovod Machine learning (Deep Learning) workloads
• Single fixed architecture using a single policy • Same scheduling for all workloads	• Increased performance for a specific workload • Unintended workloads suffer a larger performance hit

Previous Approaches

General purpose DEFs

- Apache Spark

In-memory Data intensive workloads

- Flink

Prioritises latency sensitive workloads

- Hadoop

Long running batch processes

Workload specific DEFs

- Ray

Prioritises reinforcement learning workloads

- Storm

Low-latency/real-time task computation

- Horovod

Machine learning (Deep Learning) workloads

Hybrid DEFs

- Hawk

Provides two approaches to scheduling at the cost of overall performance

- Mercury

Expands Apache spark providing hybrid scheduling while maintaining fairness

---

- Single fixed architecture using a single policy
- Same scheduling for all workloads

- Increased performance for a specific workload
- Unintended workloads suffer a larger performance hit

- Capable of scheduling a larger workload set
- Limited by the overhead of selecting the correct approach

Previous Approaches

General purpose DEFs

- Apache Spark

In-memory Data intensive workloads

- Flink

Prioritises latency sensitive workloads

- Hadoop

Long running batch processes

- Single fixed architecture using a single policy
- Same scheduling for all workloads

Workload specific DEFs

- Ray

Prioritises reinforcement learning workloads

- Storm

Low-latency/real-time task computation

- Horovod

Machine learning (Deep Learning) workloads

- Increased performance for a specific workload
- Unintended workloads suffer a larger performance hit

Hybrid DEFs

- Hawk

Provides two approaches to scheduling at the cost of overall performance

- Mercury

Expands Apache spark providing hybrid scheduling while maintaining fairness

---

- Capable of scheduling a larger workload set
- Limited by the overhead of selecting the correct approach

Adaptive/Learning DEFs

- MR Adapt

Policy adaptation using user provided completion times

- FlexTree

Model-based adaptation for workloads

- Decima

Learns a scheduling policy for a specific workload type

- Require user intervention for correct adaptation
- Learning based approaches are efficient while limited to a specific workload type

Self-Adaptive Approach

Create a DEF which is capable of adapting the scheduling policy at runtime to reduce the scheduling overhead for a current workload of a larger set.

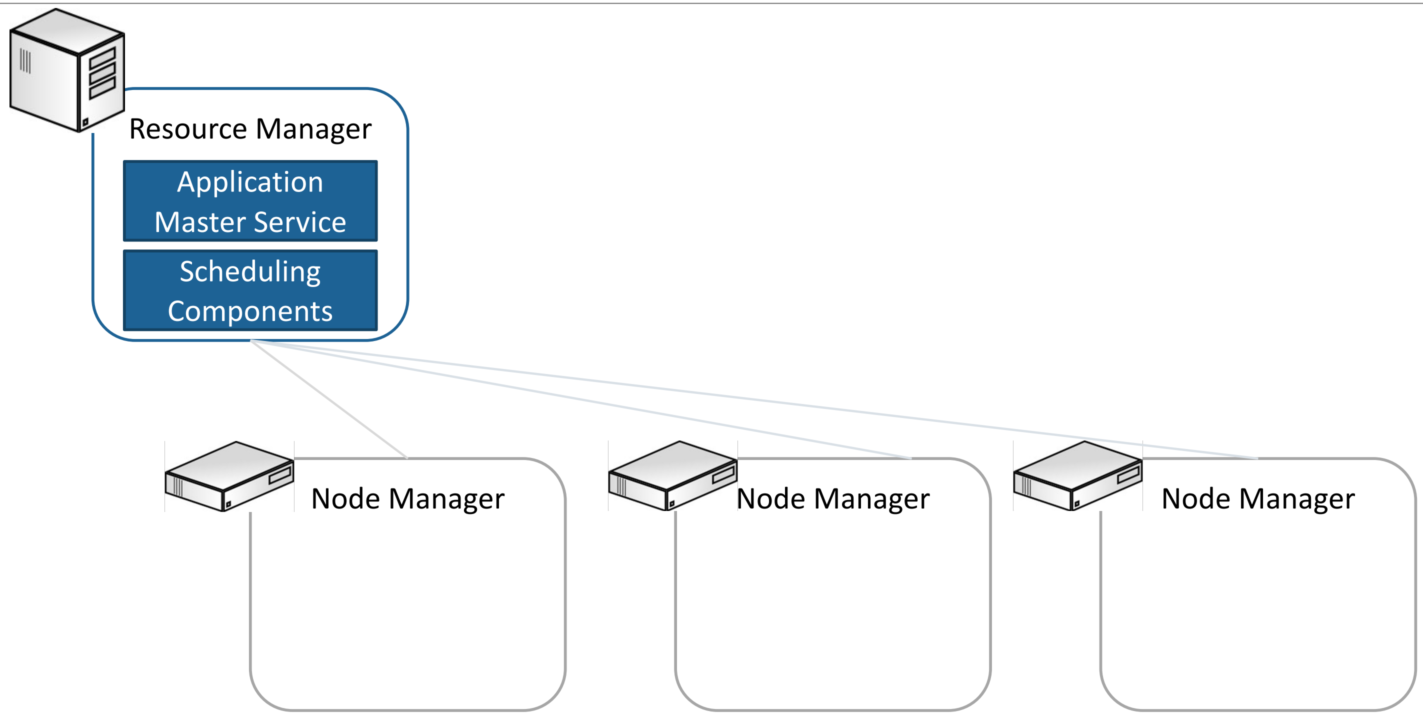
- Using the Dana programming language to create components containing DEF logic
- Components are assembled into a composition creating a complex system (node within DEF)

Self-adaptation of a scheduling policy for a given workload is challenging:

- Adaptation requires an identifiable point to change during workloads
- A machine learning agent to learn the near optimal composition for a given workload

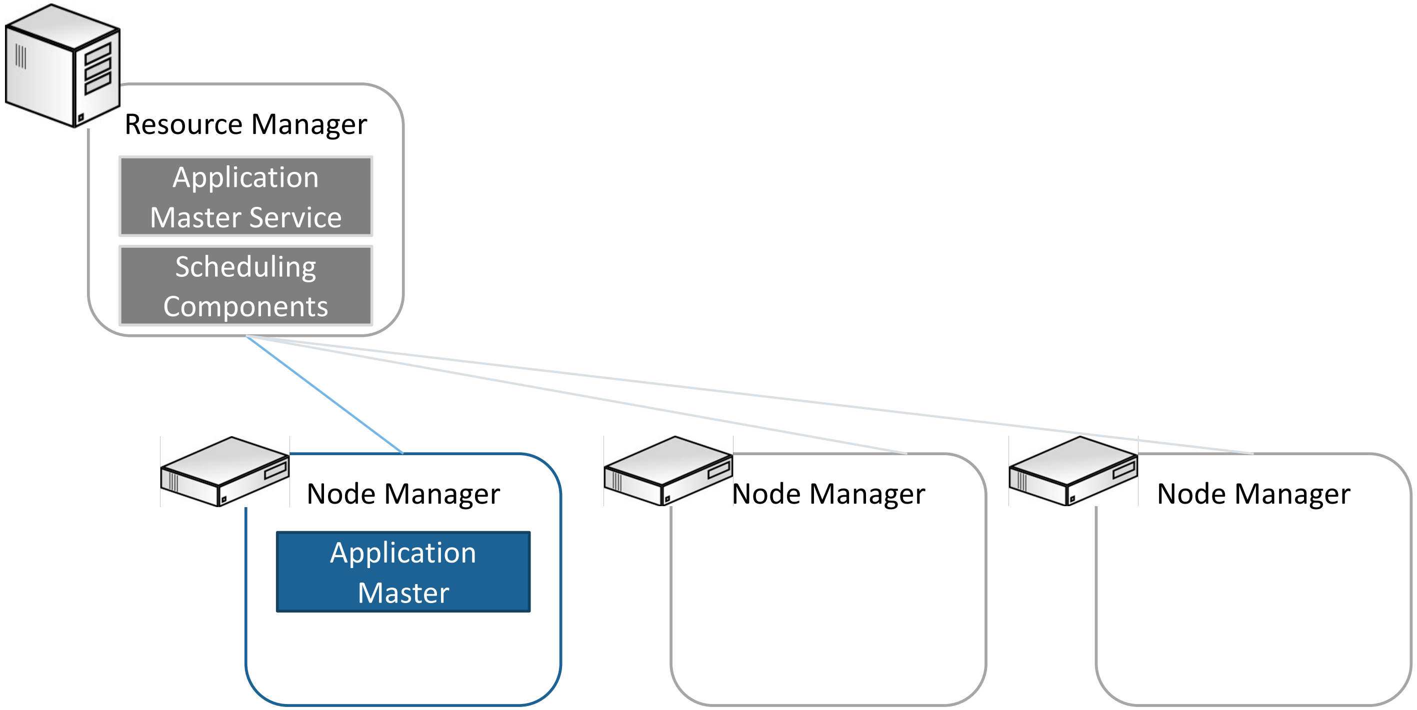
System Architecture

User submitted workload is passed to the Resource Manager starting the deployment of Application Masters



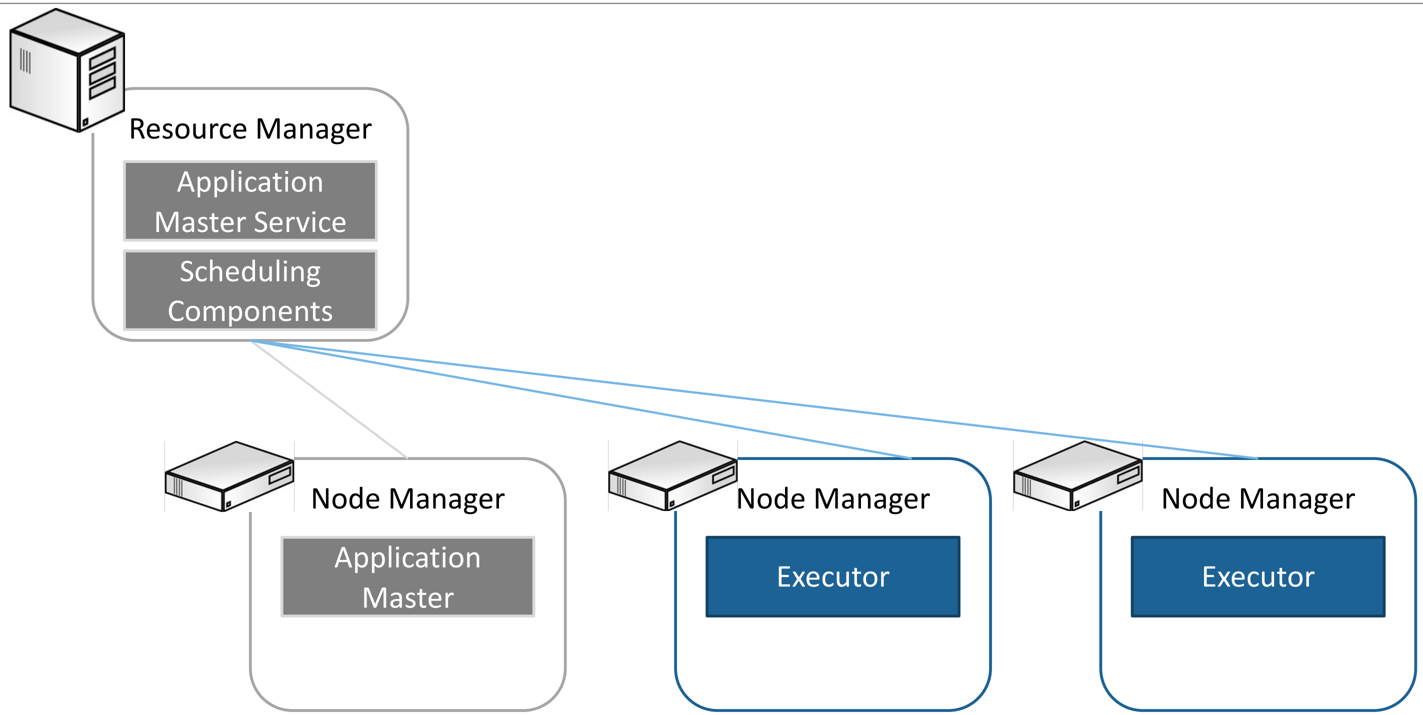
System Architecture

Application Master begins on a corresponding Node Manager and request Executors



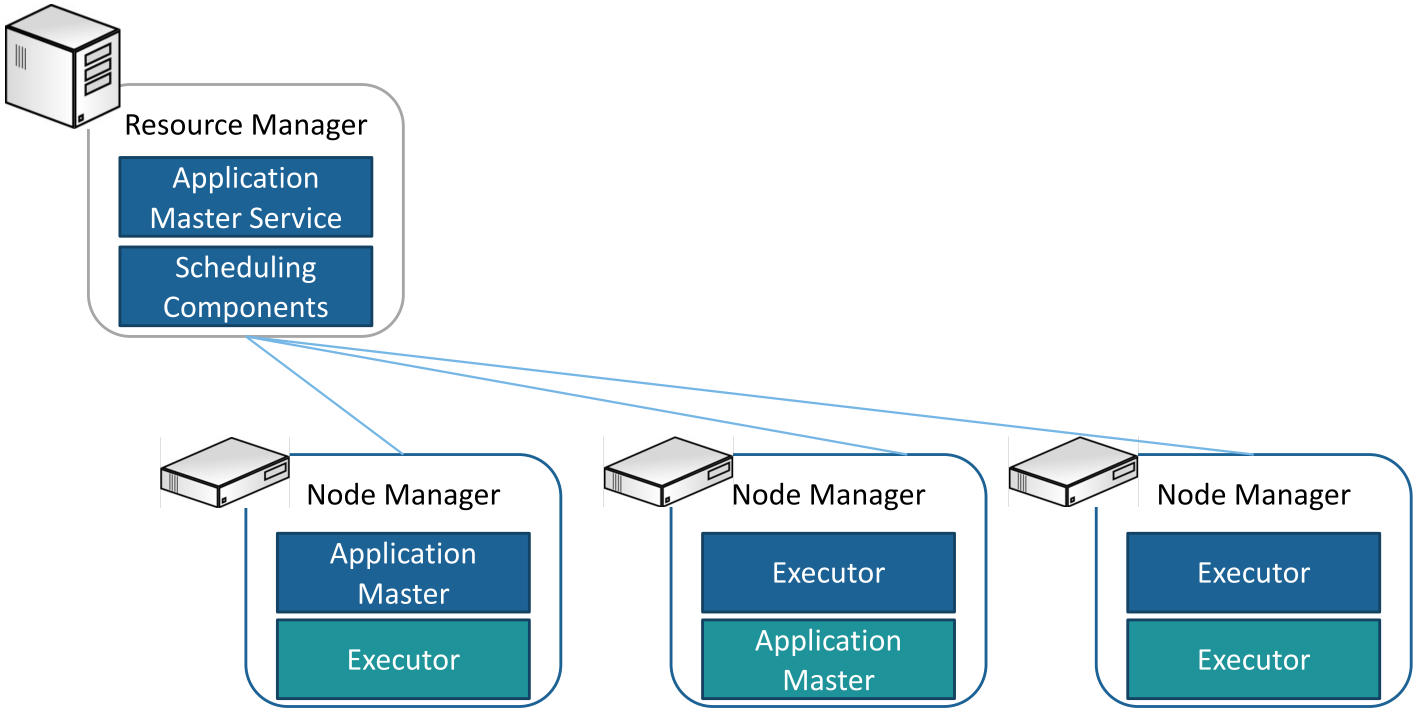
System Architecture

Executors are scheduled, deployed and begin completing computational tasks passed from the Application Master

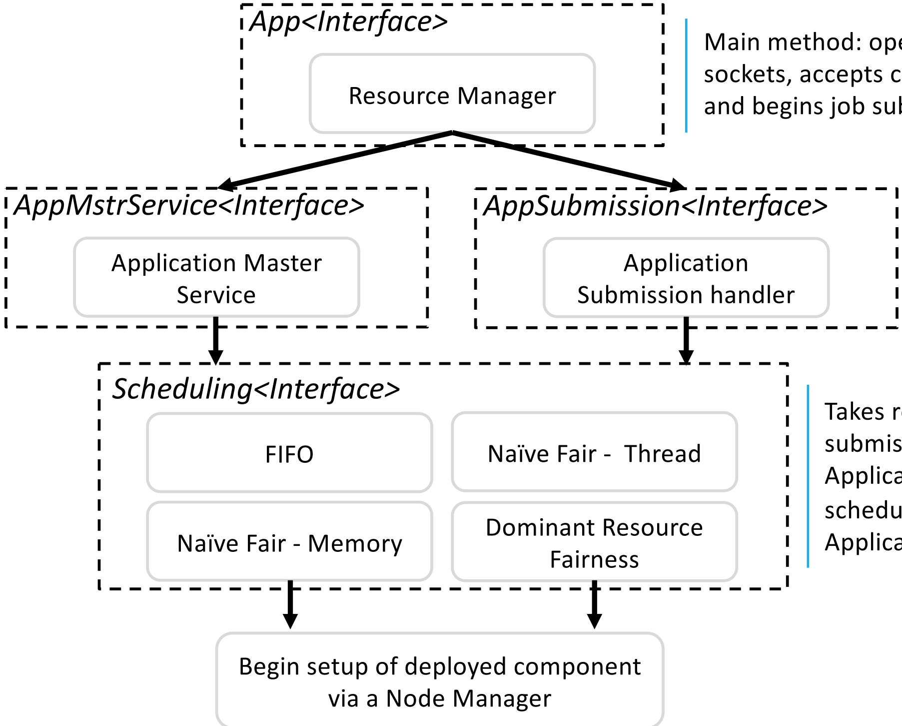


System Architecture

Previous process is repeated for the remaining submitted jobs within the schedulers queue



Emergent scheduler – example



Main method: opens server sockets, accepts client requests and begins job submission

Takes executor requests from registered Application Masters

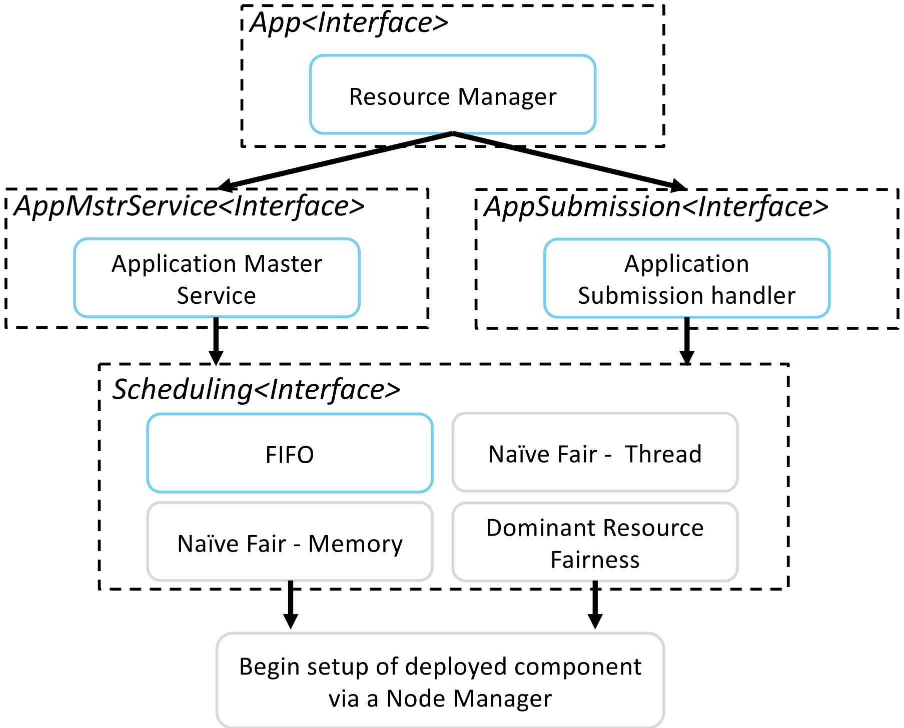
Takes user submitted applications and parses all required information

Takes requests from application submission handler and Application Master Service, scheduling and deploying Application masters and Executors

Emergent scheduler – example (continued)

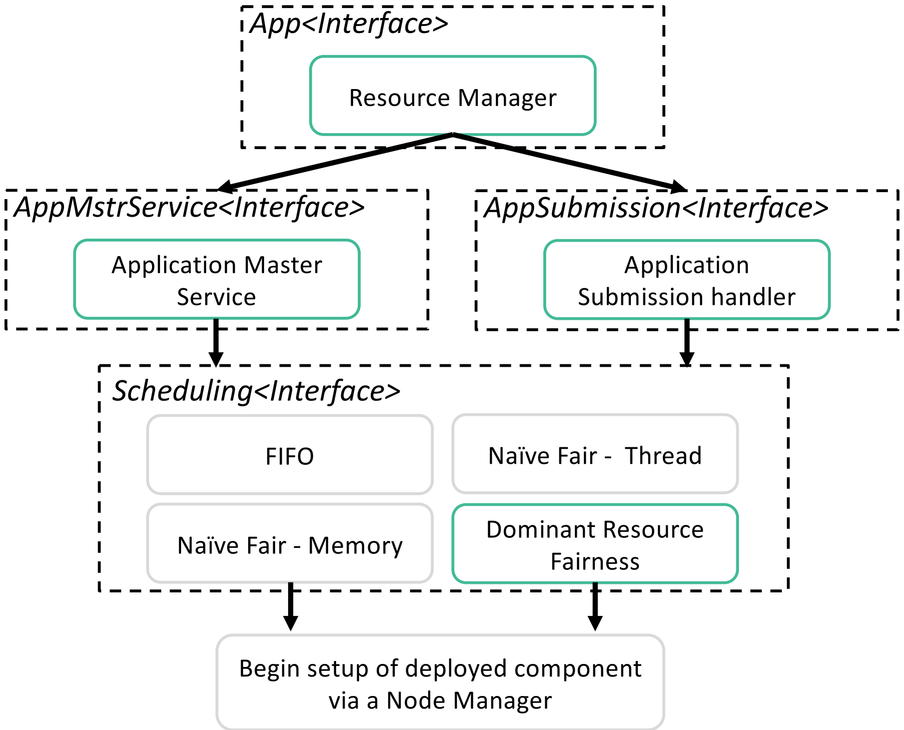
Workload A

Submission of 15 fine grained jobs



Emergent scheduler – example (continued)

Workload B
Submission of 15 coarse grained jobs



Methodology

The experiments consisted of 15 synthetic workloads of varying granularity:

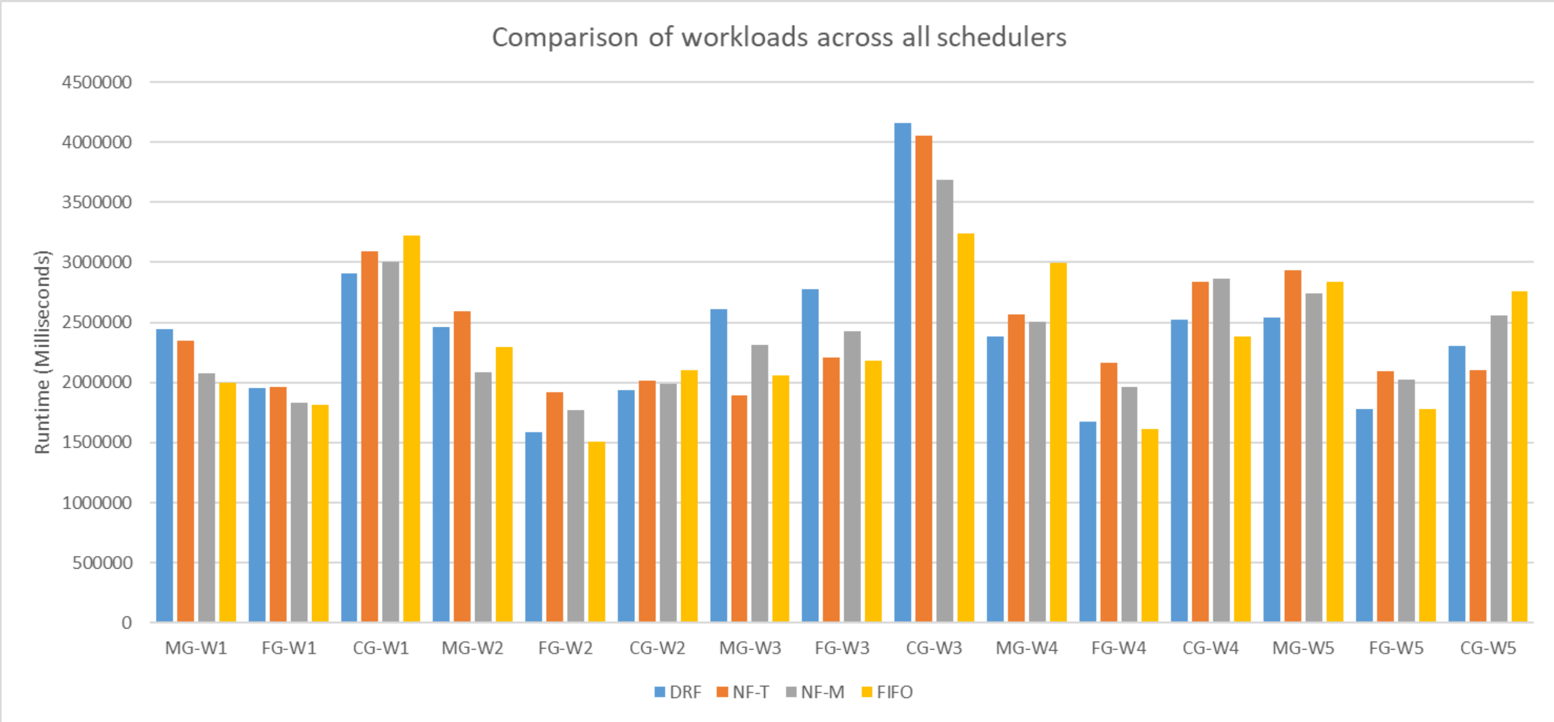
- Coarse
- Fine
- Mixed

Using 4 scheduling policies:

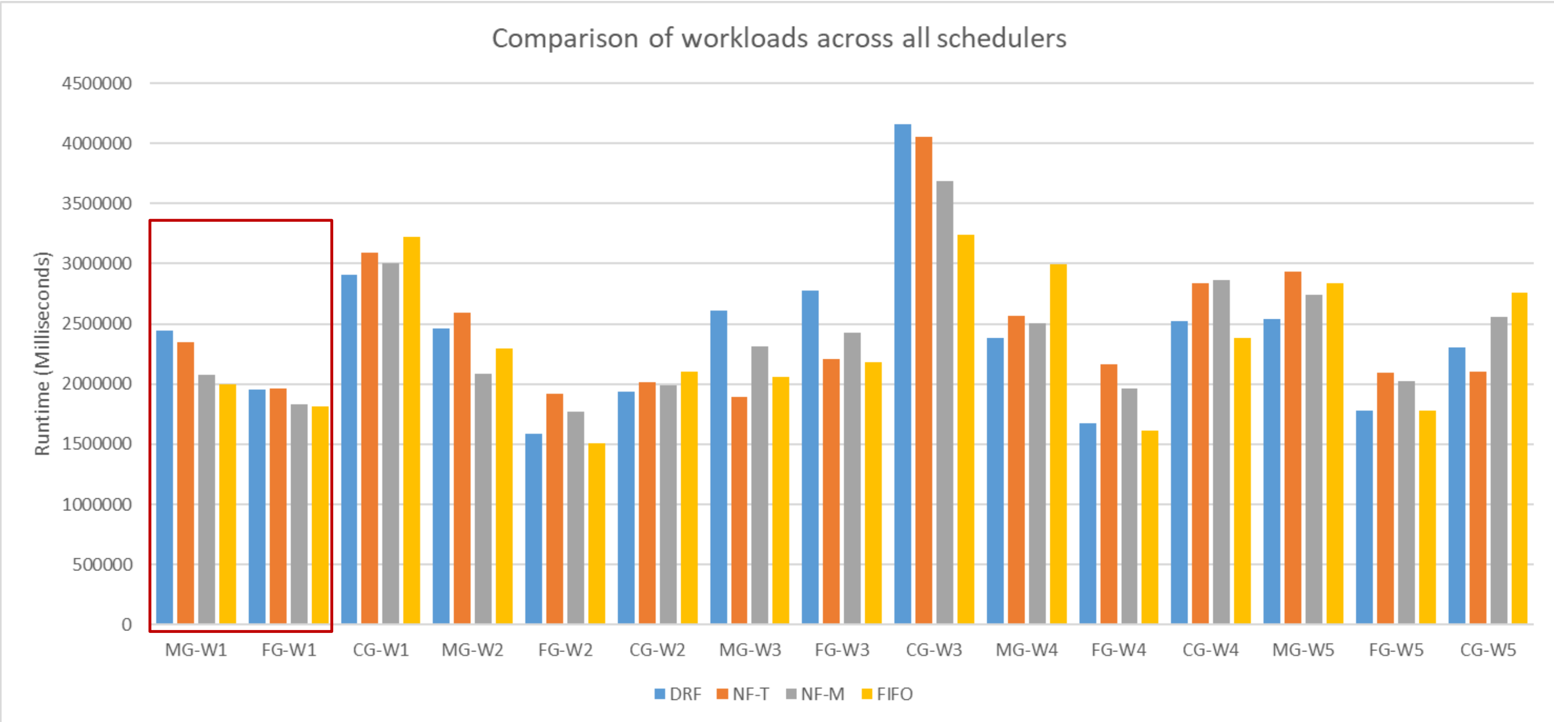
- FIFO
- Dominant resource fairness
- Naïve fair (Thread)
- Naive fair (Memory)

- The workloads were run on a cluster comprised of 5 machines:
  - 3.6GHz 8 cores
  - 16GB memory

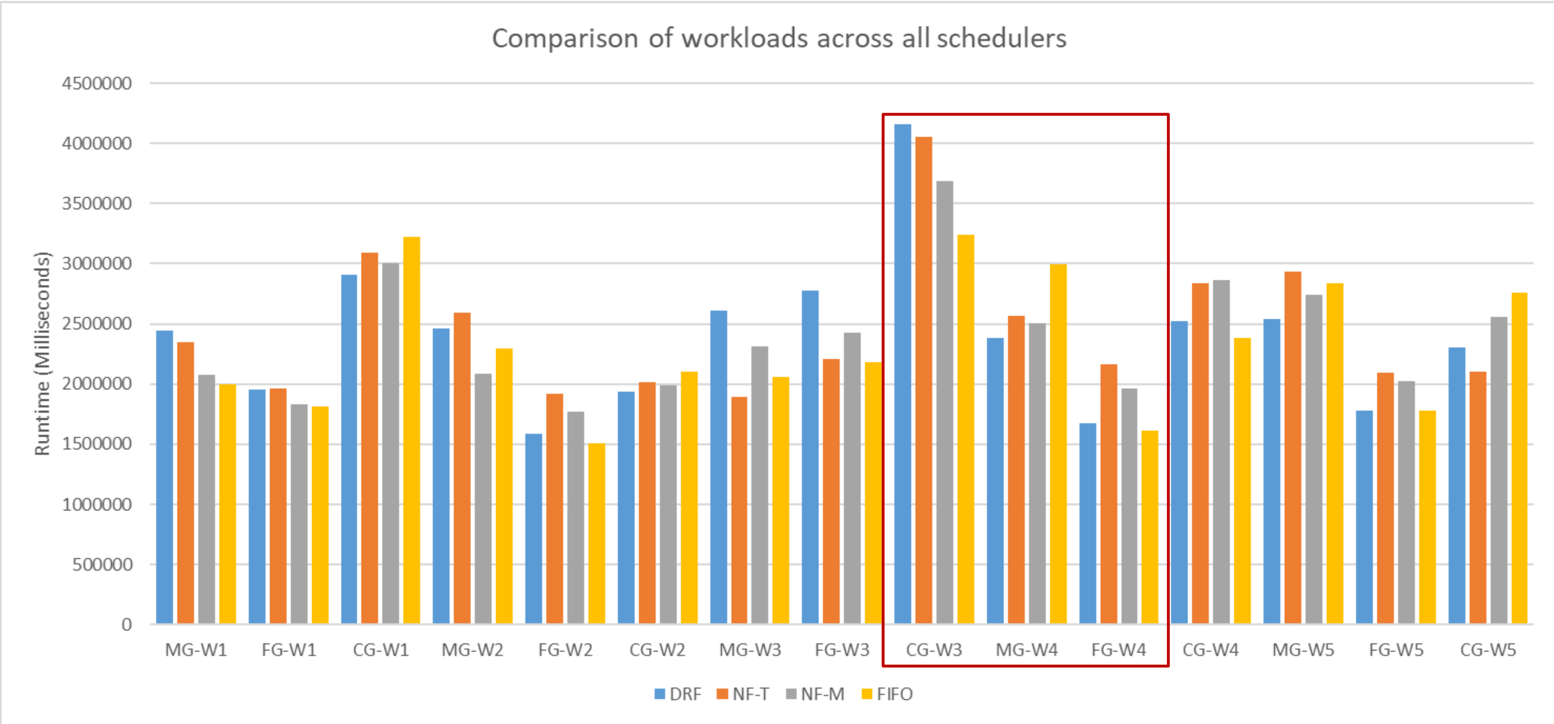
Results: Comparison for all workloads



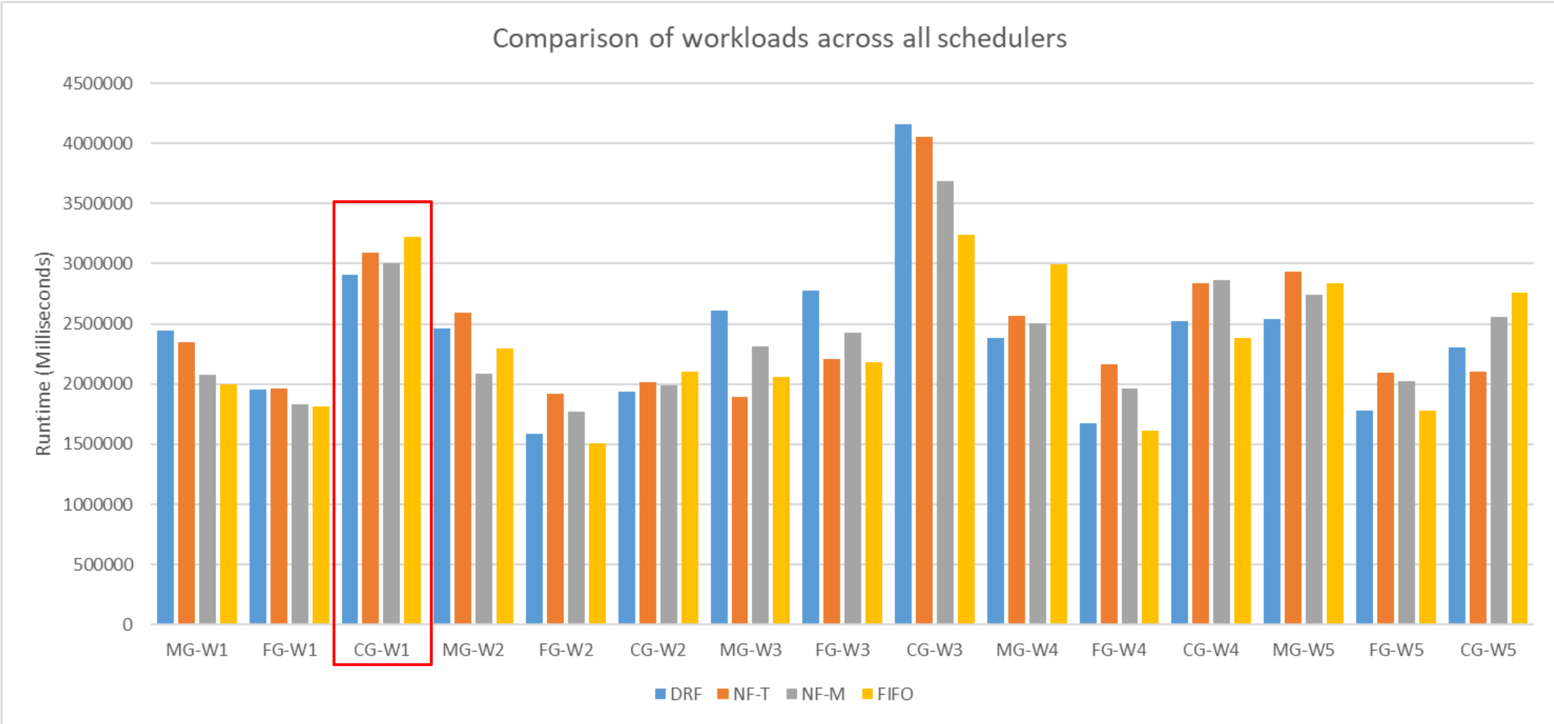
Results: Comparison for all workloads



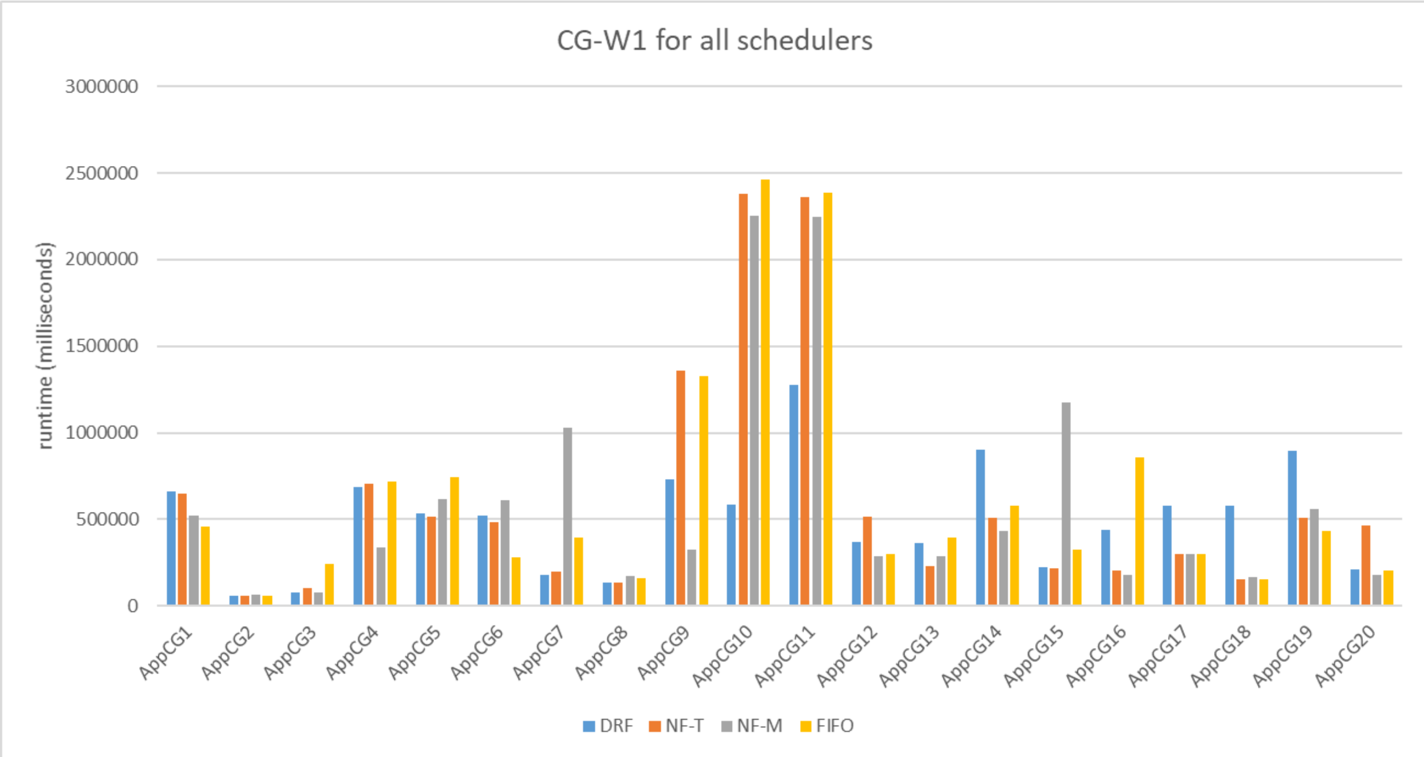
Results: Comparison for all workloads



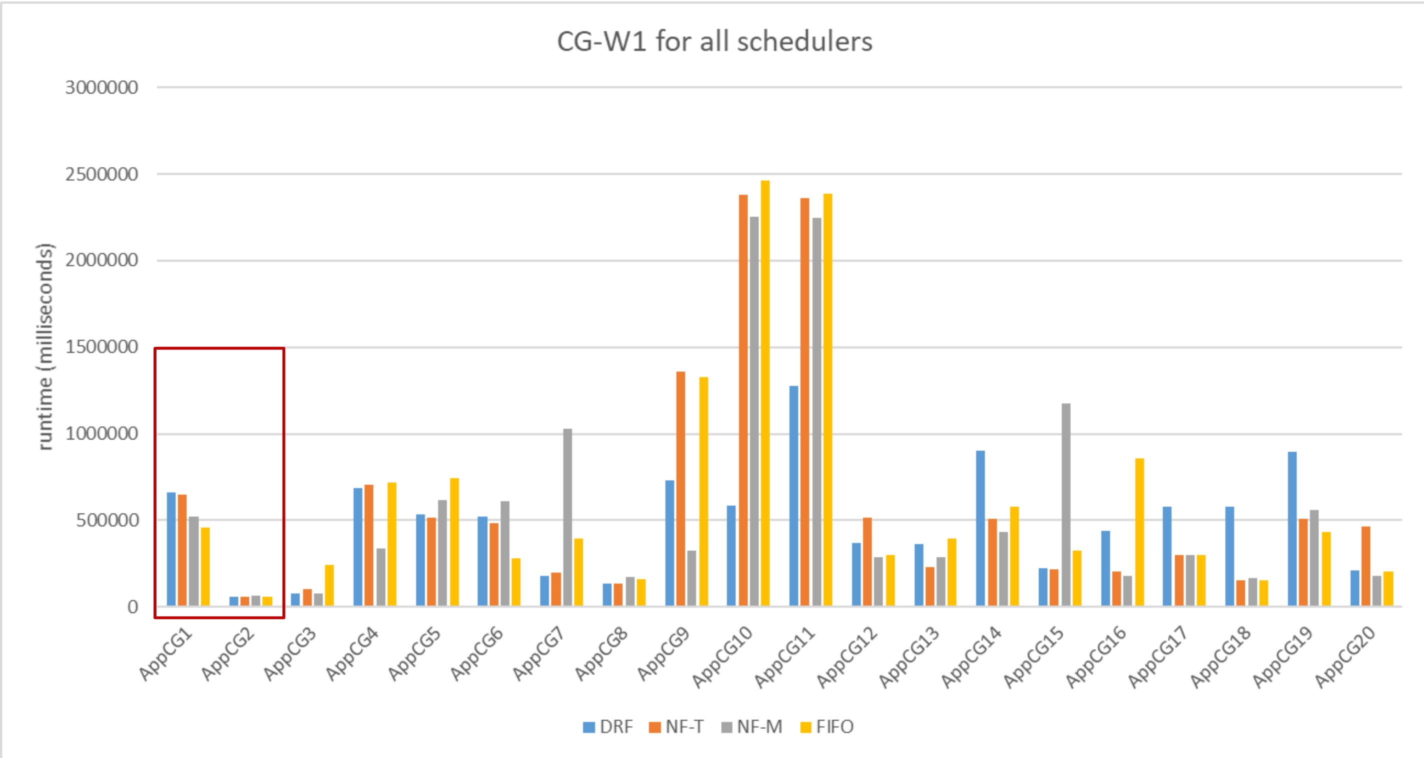
Results: Comparison for all workloads



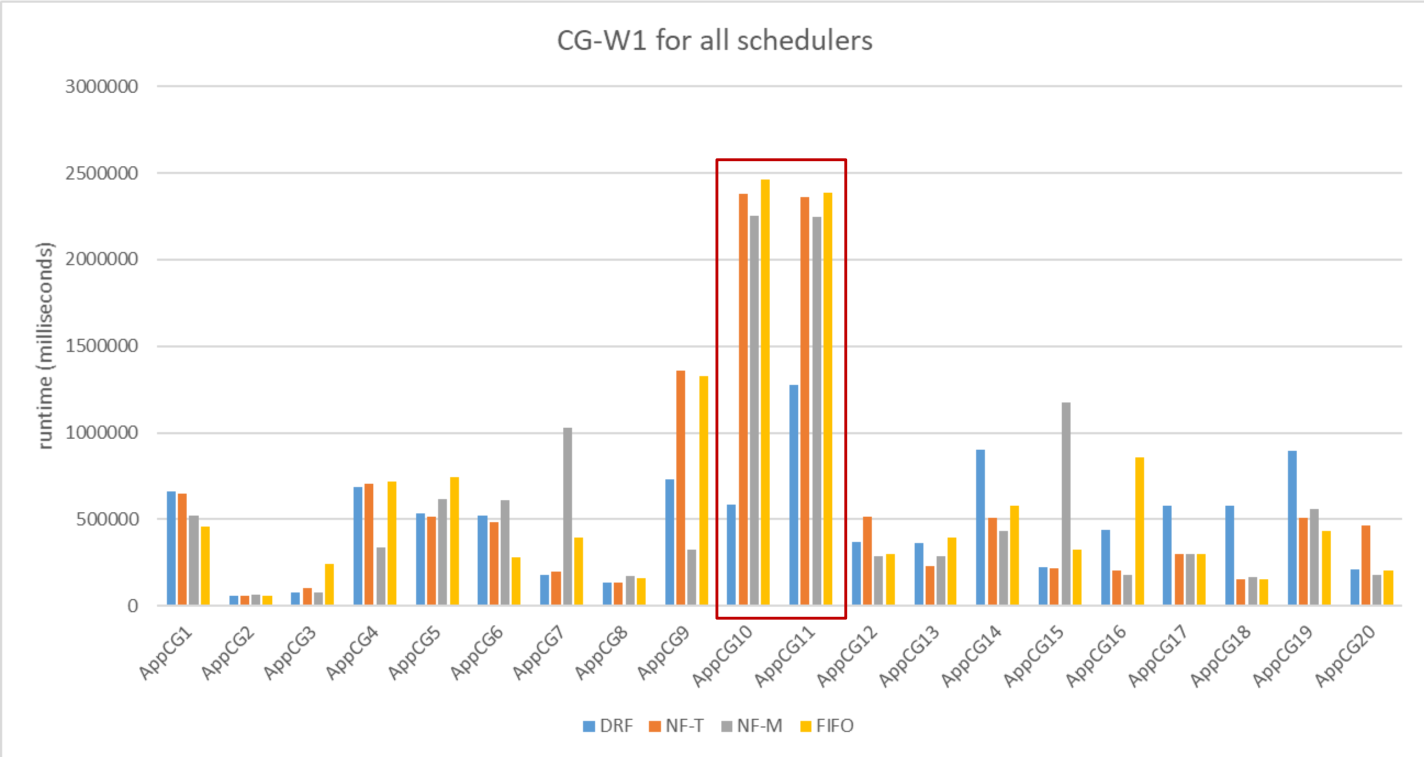
Results for workload CG-W1



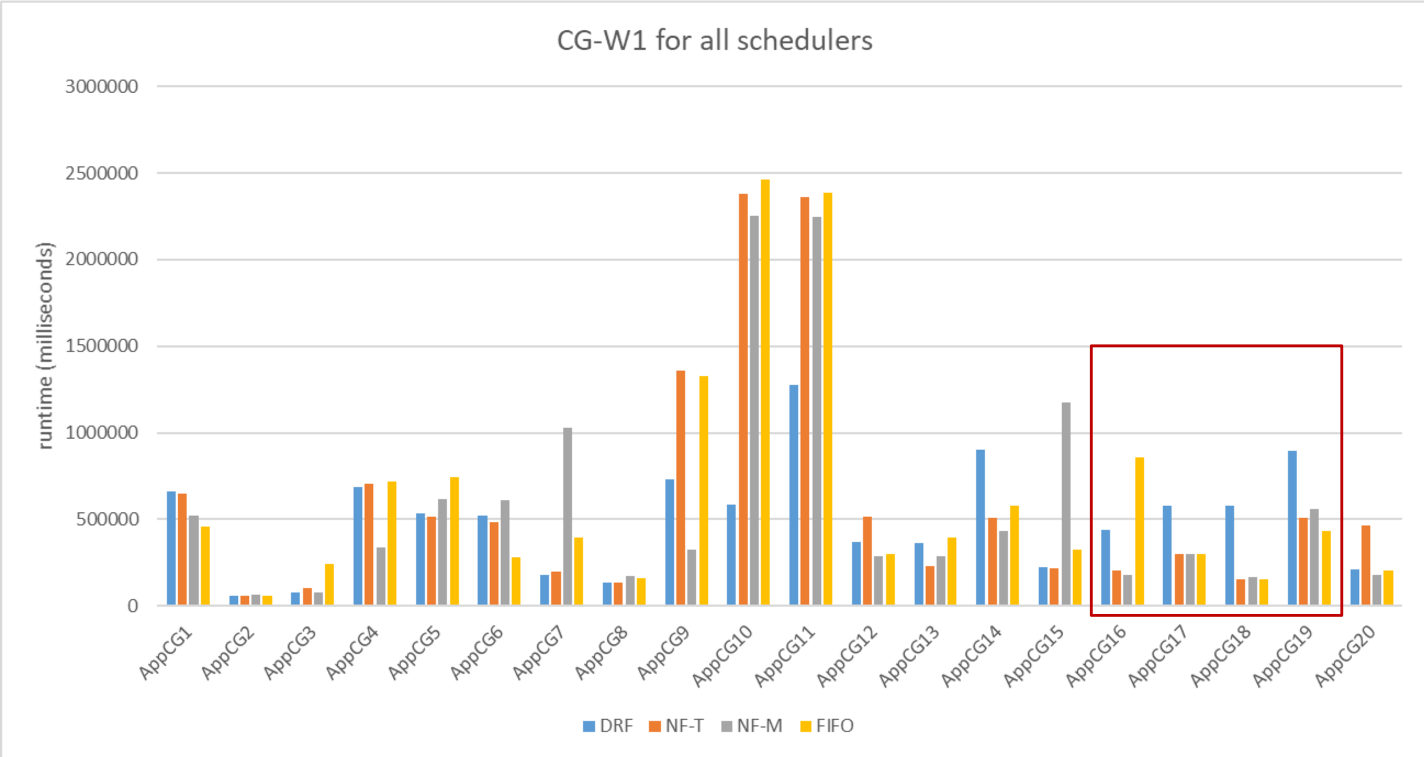
Results for workload CG-W1



Results for workload CG-W1



Results for workload CG-W1



Future Work

The previously shown results identify points within the batched and individual workload traces where a performance gain may be obtained through adaptation.

- Experiment and compare public benchmarks/workloads of the same type
- Explore the efficiency of machine learning agents for adaptation
- Compare a self-adaptive scheduling approach