USING SIMPLE PID CONTROLLERS TO PREVENT AND MITIGATE FAULTS IN SCIENTIFIC WORKFLOWS

Rafael Ferreira da Silva[1], Rosa Filgueira[2], Ewa Deelman[1],
Erola Pairo-Castineira[3], Ian Michael Overton[4], Malcolm Atkinson[5]

[1]USC Information Sciences Institute
[2]British Geological Survey, Lyell Centre
[3]MRC Institute of Genetics and Molecular Medicine, University of Edinburgh
[4]Usher Institute of Population Health Sciences and Informatics, University of Edinburgh
[5]School of Informatics, University of Edinburgh

11th Workflows in Support of Large-Scale Science (WORKS'16)
Salt Lake City, UT – November 14th, 2016

USC Viterbi
School of Engineering
Information Sciences Institute

OUTLINE

Introduction

Scientific Workflows
Motivation
Related Work

PID Controllers

Definition
Control System Loop

Defining Controllers

Data Management
Memory Management

Experimental Evaluation

Workflow Application
Experiments Conditions
Results and Discussion

Tuning PID Controllers

Ziegler-Nichols Method
Experimental Evaluation

Summary

Conclusions
Future Research Directions

WHY SCIENTIFIC WORKFLOWS?

Automation

Enables parallel, distributed computations

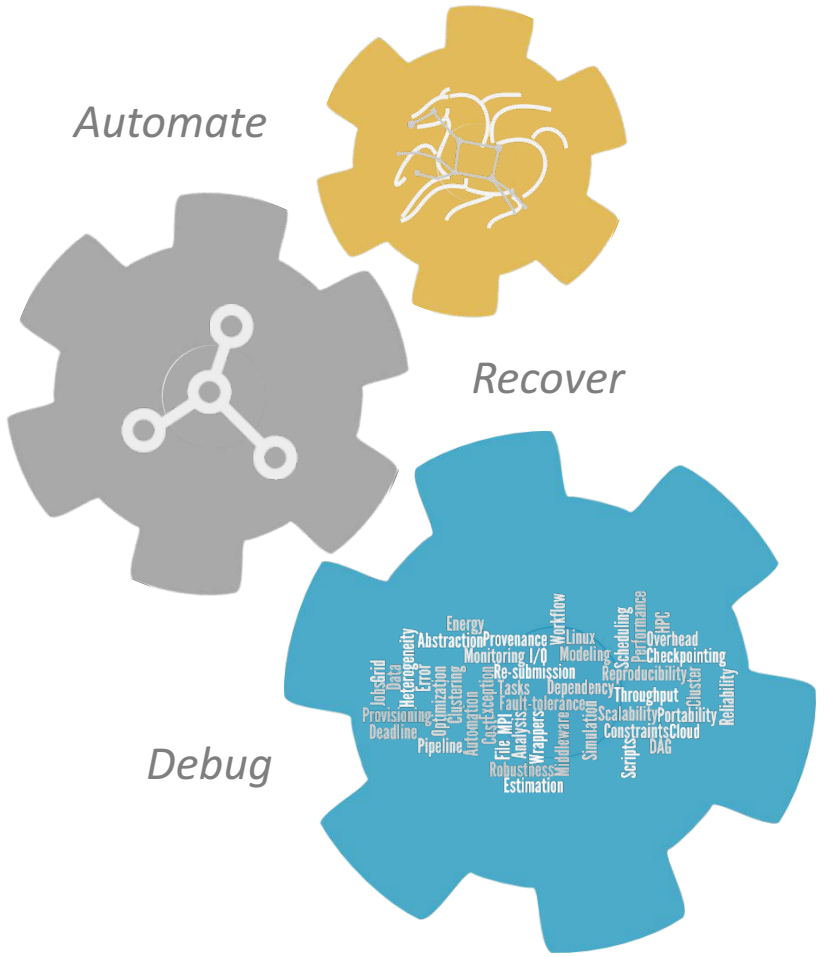
Automatically executes data transfers

Recover & Debug

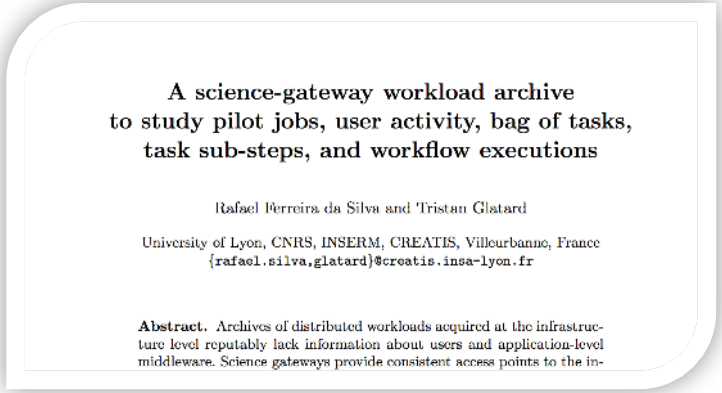
Handles failures with to provide reliability
Keeps track of data and files

Reproducibility

Reusable, aids reproducibility
Records how data was produced (provenance)



MOTIVATION



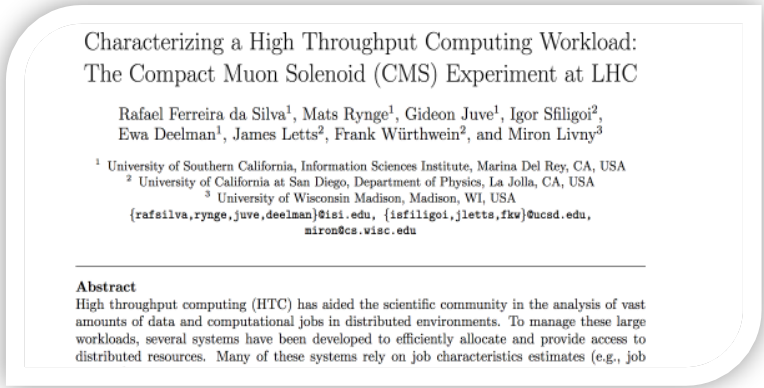
A science-gateway workload archive to study pilot jobs, user activity, bag of tasks, task sub-steps, and workflow executions

Rafael Ferreira da Silva and Tristan Glatard

University of Lyon, CNRS, INSERM, CREATIS, Villeurbanne, France
{rafael.silva,glatard}@creatis.insa-lyon.fr

Abstract. Archives of distributed workloads acquired at the infrastructure level reputably lack information about users and application-level middleware. Science gateways provide consistent access points to the in-

Grid computing
180k failed tasks out of 340k



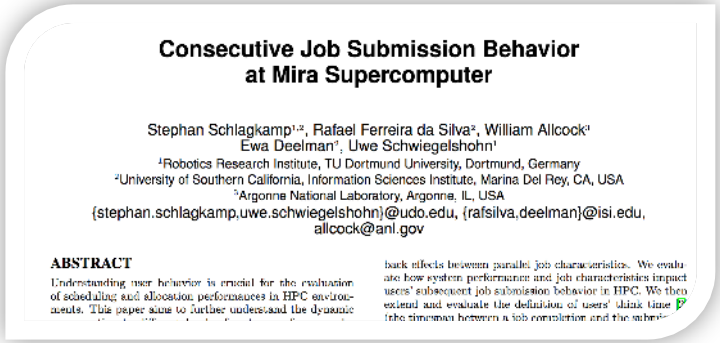
Characterizing a High Throughput Computing Workload: The Compact Muon Solenoid (CMS) Experiment at LHC

Rafael Ferreira da Silva[1], Mats Rynge[1], Gideon Juve[1], Igor Sfiligoi[2], Ewa Deelman[1], James Letts[2], Frank Würthwein[2], and Miron Livny[3]

[1] University of Southern California, Information Sciences Institute, Marina Del Rey, CA, USA
[2] University of California at San Diego, Department of Physics, La Jolla, CA, USA
[3] University of Wisconsin Madison, Madison, WI, USA
{rafsilva,rynge,juve,deelman}@isi.edu, {isfiligoi,jletts,fkw}@ucsd.edu, miron@cs.wisc.edu

Abstract
High throughput computing (HTC) has aided the scientific community in the analysis of vast amounts of data and computational jobs in distributed environments. To manage these large workloads, several systems have been developed to efficiently allocate and provide access to distributed resources. Many of these systems rely on job characteristics estimates (e.g., job

CMS (Aug 2014)
385k failed tasks out of 790k



Consecutive Job Submission Behavior at Mira Supercomputer

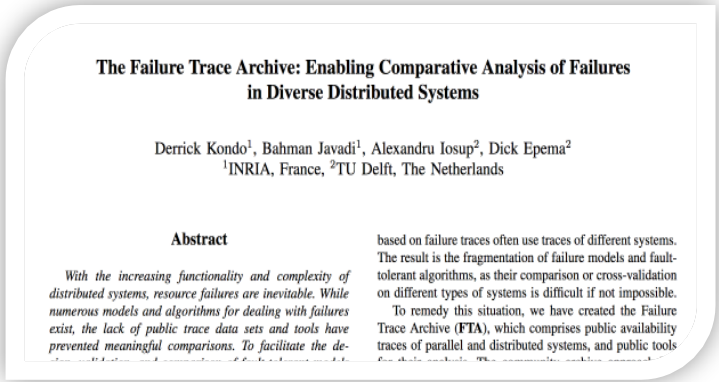
Stephan Schlagkamp[1,2], Rafael Ferreira da Silva[2], William Allcock[3]
Ewa Deelman[2], Uwe Schwiegelshohn[1]
[1]Robotics Research Institute, TU Dortmund University, Dortmund, Germany
[2]University of Southern California, Information Sciences Institute, Marina Del Rey, CA, USA
[3]Argonne National Laboratory, Argonne, IL, USA
{stephan.schlagkamp,uwe.schwiegelshohn}@udo.edu, {rafsilva,deelman}@isi.edu, allcock@anl.gov

ABSTRACT
Understanding user behavior is crucial for the evaluation of scheduling and allocation performances in HPC environments. This paper aims to further understand the dynamic

back effects between parallel job characteristics. We evaluate how system performance and job characteristics impact users' subsequent job submission behavior in HPC. We then extend and evaluate the definition of users' think time [the timespan between a job completion and the subm

Mira (2014)
over 14k failed jobs out of 80k

The Failure Trace Archive
26 datasets from 2006-2014



The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems

Derrick Kondo[1], Bahman Javadi[1], Alexandru Iosup[2], Dick Epema[2]
[1]INRIA, France, [2]TU Delft, The Netherlands

Abstract

With the increasing functionality and complexity of distributed systems, resource failures are inevitable. While numerous models and algorithms for dealing with failures exist, the lack of public trace data sets and tools have prevented meaningful comparisons. To facilitate the de-

based on failure traces often use traces of different systems. The result is the fragmentation of failure models and fault-tolerant algorithms, as their comparison or cross-validation on different types of systems is difficult if not impossible.

To remedy this situation, we have created the Failure Trace Archive (**FTA**), which comprises public availability traces of parallel and distributed systems, and public tools

SOME APPROACHES TO HANDLE FAULTS

Typical Approaches

Task Retries
Task Resubmission
Task Clustering
Checkpointing
Provenance
…

Statistical and Machine Learning

Linear Regression
Neural Networks
Classification Algorithms
Tree-based Methods
Support Vector Machines
…

Analytical Solutions

Failure Modeling
Markov Chains
Principal Component Analysis
Histograms
…

Others

Exception Handling
Game Theory
…

… AND SOME OF THEIR LIMITATIONS

> ***Most of them make strong assumptions about resource and application characteristics***
> Accurate estimates of such requirements are still a steep challenge

> ***Some approaches may overload the execution platform***

> ***Most of the systems do not prevent faults, but mitigate them***

> ***Some approaches are tied to a small set of applications***

> *We seek for an approach to **predict**, **prevent**, and **mitigate** failures in end-to-end workflow executions across distributed systems under **online** and **unknown** conditions*

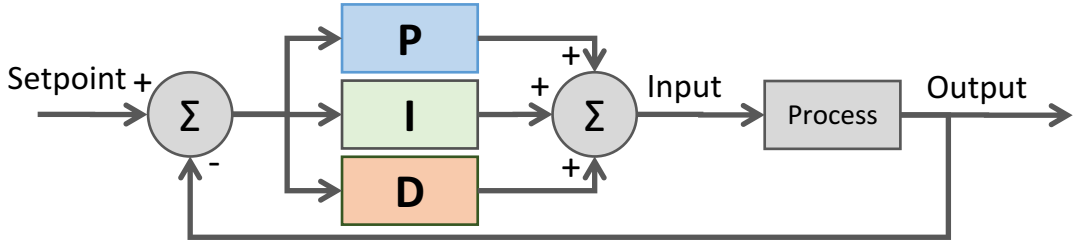
<<<

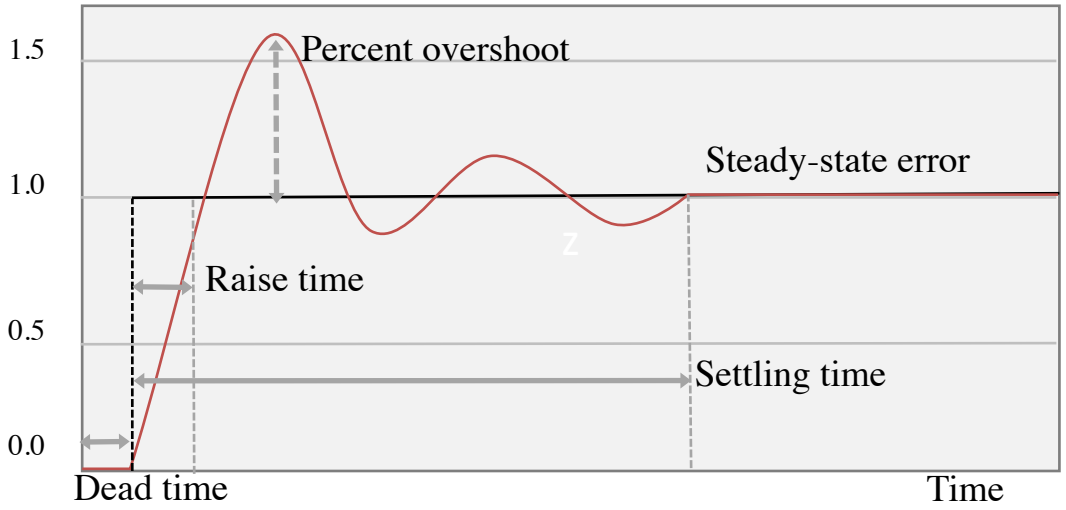
PID CONTROLLERS

Proportional-Integral-Derivative Controller

Control loop mechanism
Widely used in industrial control systems
- Temperature
- Pressure
- Flow rate
- etc.

PID controller aims at detecting the possibility of a fault far enough in advance so that an action can be performed to prevent it from happening





PROCESS VARIABLES

PID

Proportional-Integral-Derivative Controller

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

K_p: Proportional gain constant
K_i: Integral gain constant
K_d: Derivative gain constant
***e*:** error defined as the difference between the *setpoint* and the process variable value

proportional
Present error

integral
Accumulation of past errors

derivative
Prediction of future errors based on current rate of change

DATA FOOTPRINT AND MANAGEMENT

A run of scientific workflows that manipulate large data sets may lead the system to an **out of disk space fault**

PID Controller

P: the error between the *setpoint*, and the actual used disk space

I: cumulative value of the proportional responses

D: the difference between the current and the previous disk overflow (or underutilization) error values

Actions

u(t) < 0: data cleanup is used to remove unused data; or tasks are preempted

u(t) > 0: the number of concurrent task executions may be increased

MEMORY USAGE AND MANAGEMENT

The performance of **memory-intensive** operations are often limited by the **memory capacity** of the resource where the application is being executed.

PID Controller

P: error between the *setpoint* value, and the actual memory usage

I: cumulative value of previous memory usage errors

D: difference between the current and the previous memory overflow (or underutilization) error values

Actions

u(t) < 0: tasks are preempted to prevent the system to run out of memory

u(t) > 0: the WMS may spawn additional tasks for concurrent execution

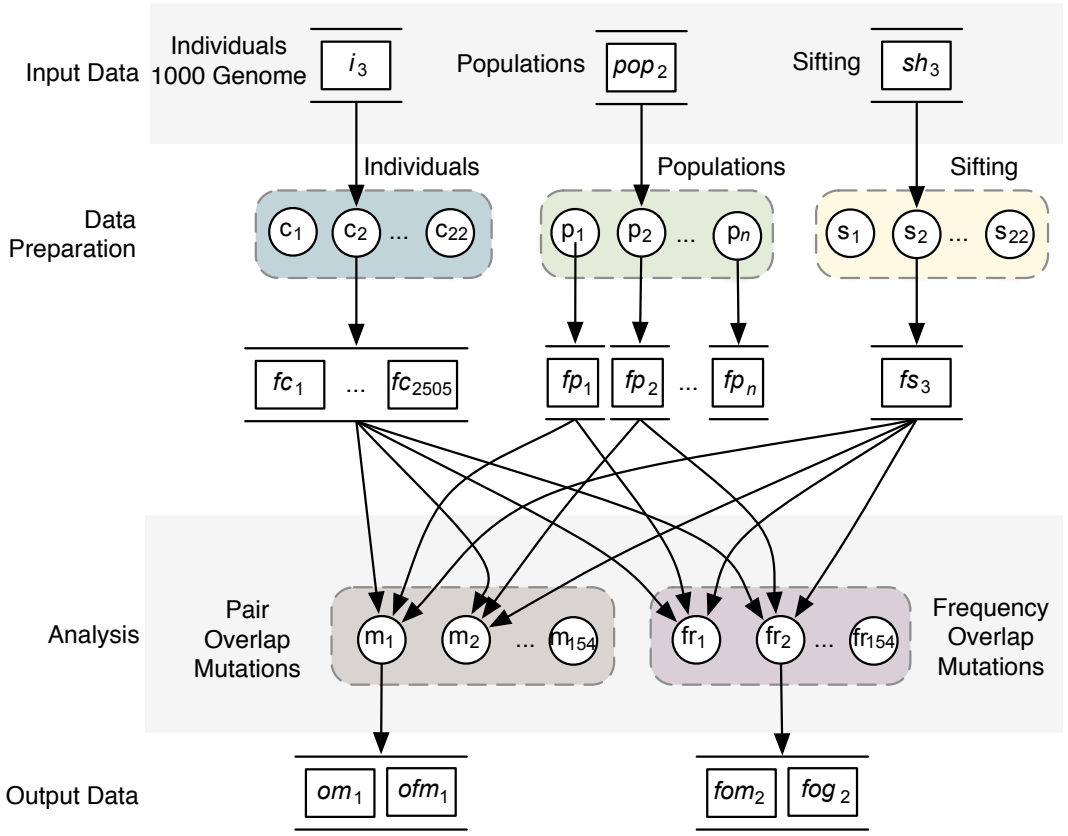
WORKFLOW APPLICATION

1000 Genome Sequencing Analysis Workflow

Identifies mutational overlaps using data from the 1000 genomes project

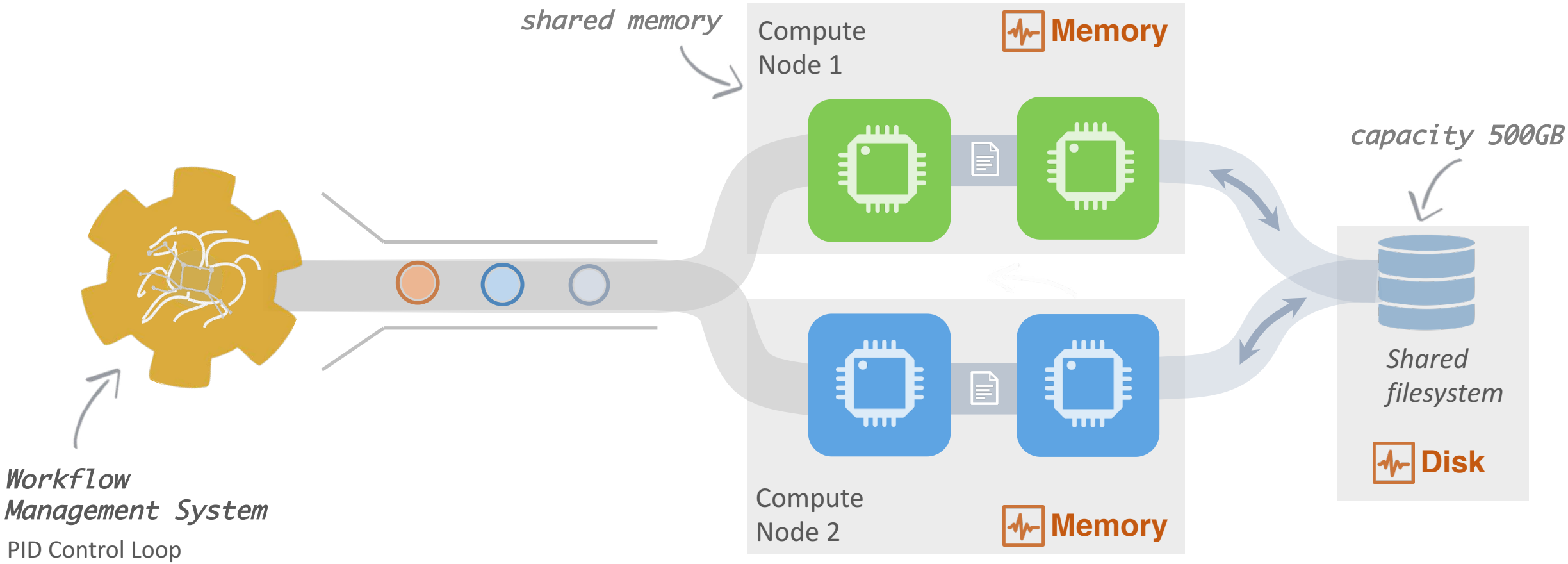
22 Individual tasks, 7 Population tasks, 22 Sifting tasks, 154 Pair Overlap Mutations tasks, and 154 Frequency Overlap Mutations tasks (**Total 359 tasks**)

*The workflow consumes/produces over **4.4TB of data**, and requires over **24TB of memory***



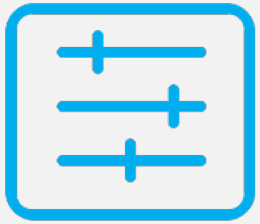
https://github.com/pegasus-isi/1000genome-workflow

EXPERIMENT SETUP



EXPERIMENT CONDITIONS

we arbitrarily define our *setpoint* as 80% of the maximum total capacity (for both storage and memory usage, and a steady-state error of 5%



We assume $K_p = K_i = K_d = 1$ (no tuning)

Execution are performed under **online** and **unknown conditions**

Reference Workflow Execution

Computed offline under known conditions

Averaged Makespan: ~106h

(standard deviation < 5%)

the decision on the number of tasks to be **scheduled or preempted** is computed as the ***min*** between the response value of the unique disk usage PID controller, and the memory PID controller per resource

OVERALL MAKESPAN EVALUATION

EVALUATION

Average workflow makespan for different configurations of the controllers

Proportional

$K_p = 1,\ K_i = K_d = 0$

Makespan: 138.76h
Slowdown: 1.30

only mitigates faults

Proportional-Integral

$K_p = K_i = 1,\ K_d = 0$

Makespan: 126.69h
Slowdown: 1.19

Proportional-Integral-Derivative

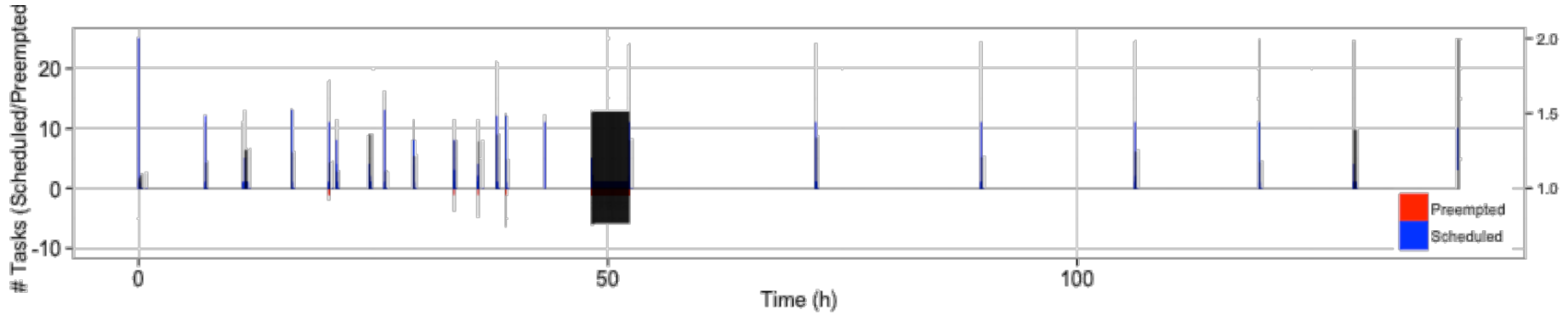
$K_p = K_i = K_d = 1$

Makespan: 114.96h
Slowdown: 1.08

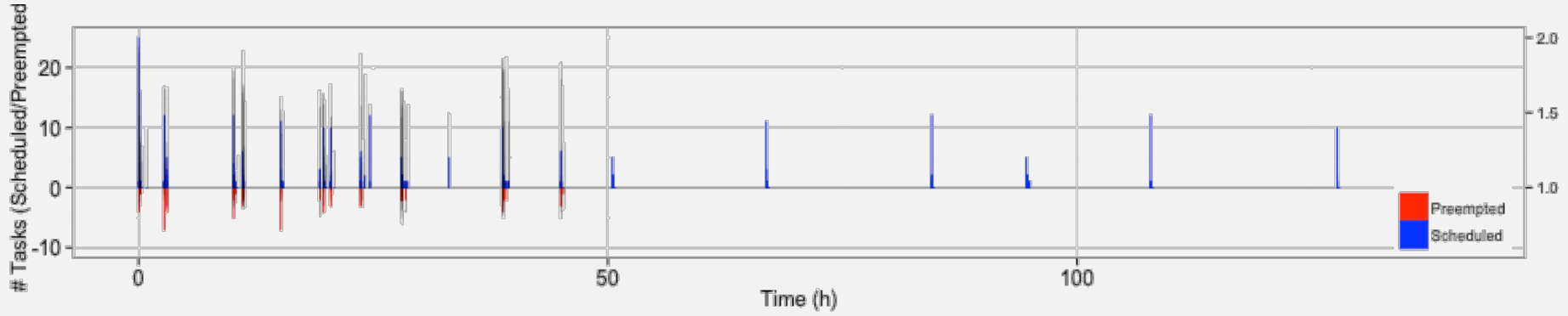
prevents faults

EXPERIMENTS: DATA FOOTPRINT

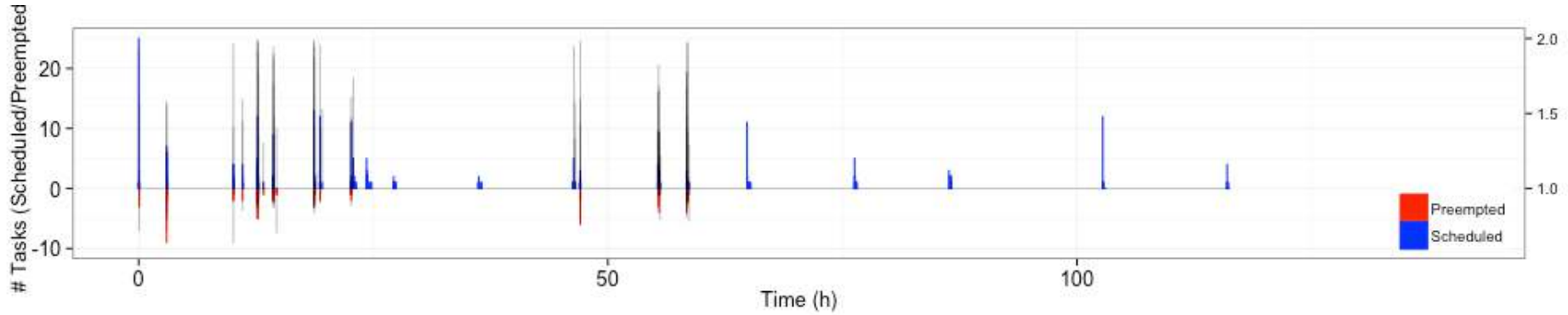
PROPORTIONAL



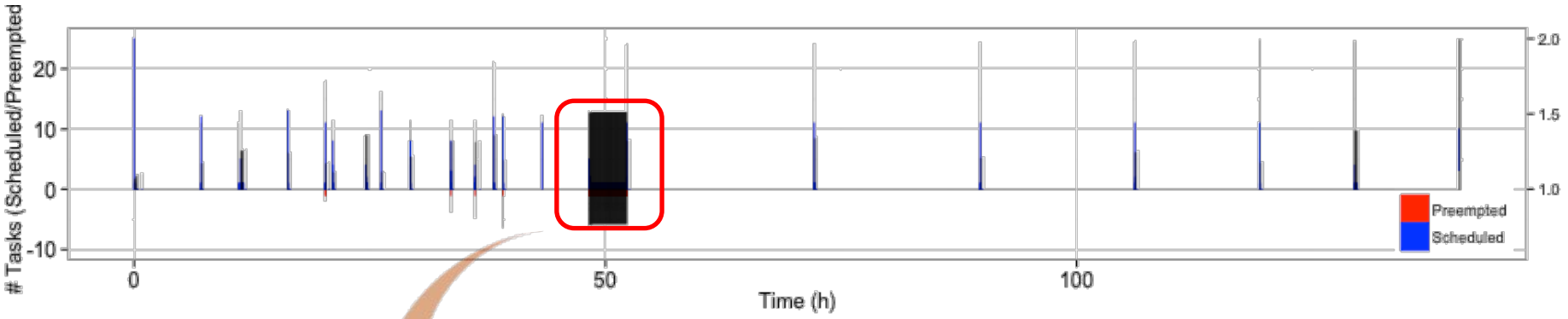
PROPORTIONAL INTEGRAL



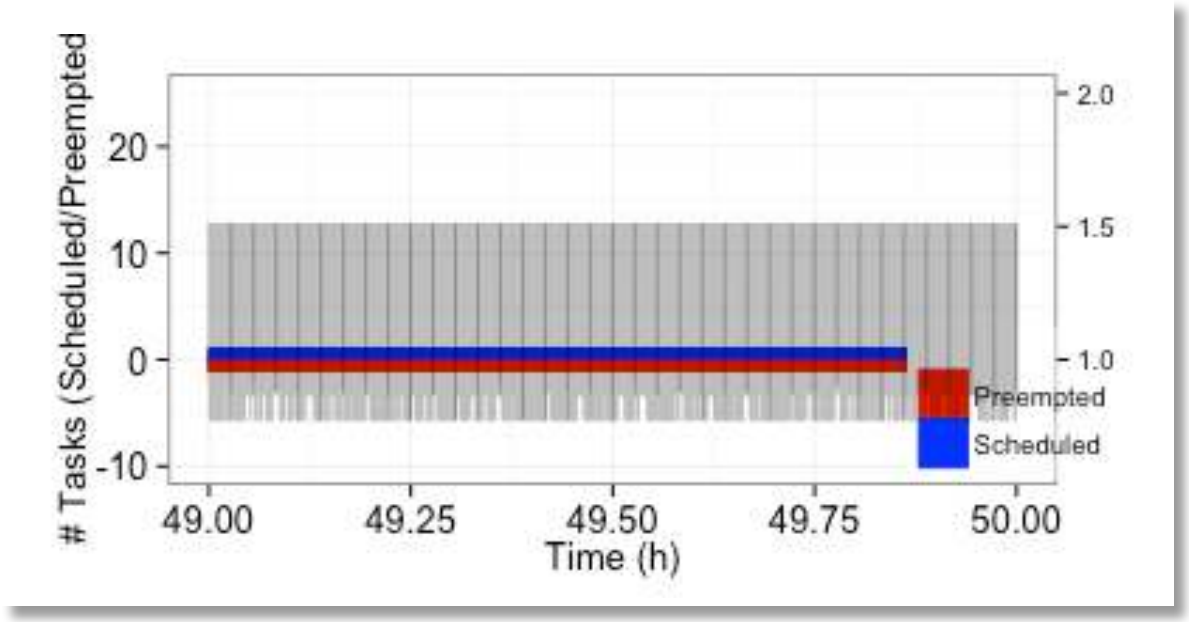
PROPORTIONAL INTEGRAL DERIVATIVE



EXPERIMENTS: DATA FOOTPRINT

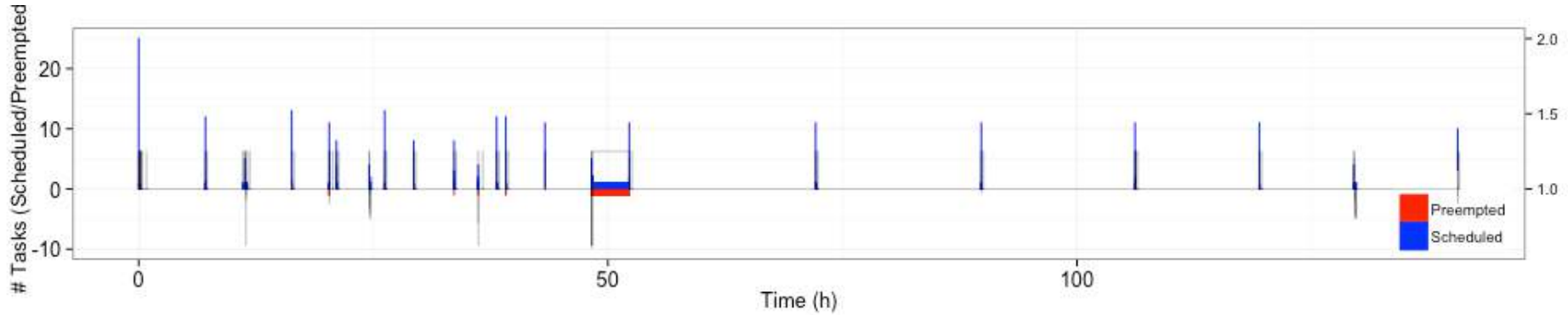


*This process occurs at about **4h**, and performs more than **6,000 preemptions***

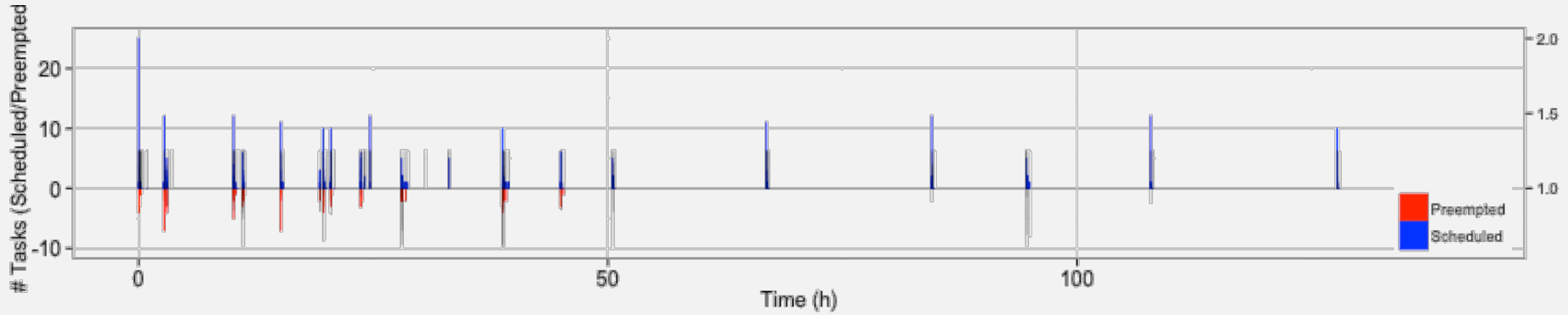


EXPERIMENTS: MEMORY USAGE

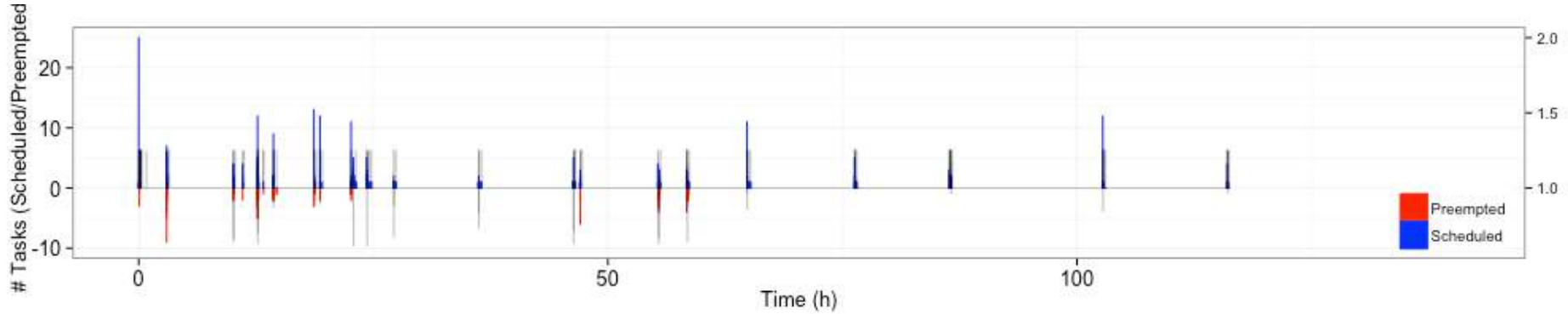
PROPORTIONAL



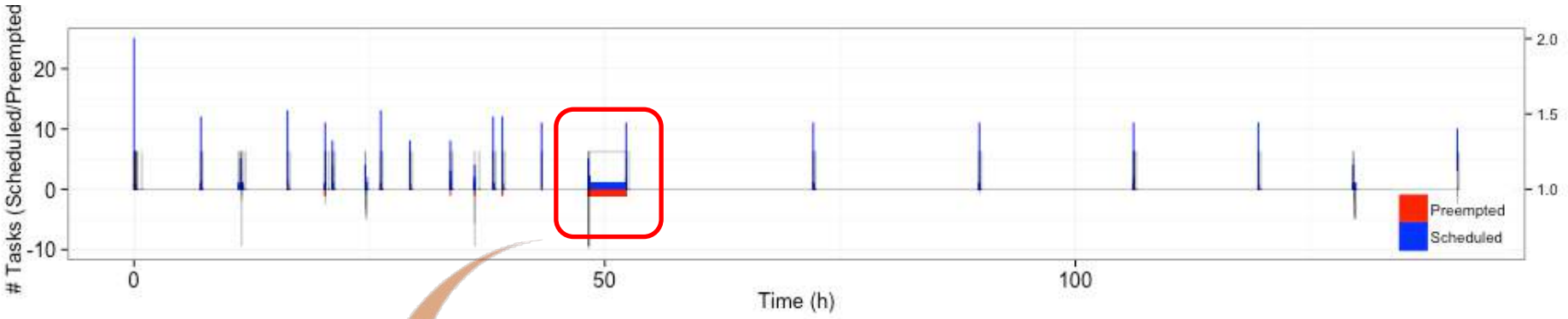
PROPORTIONAL INTEGRAL



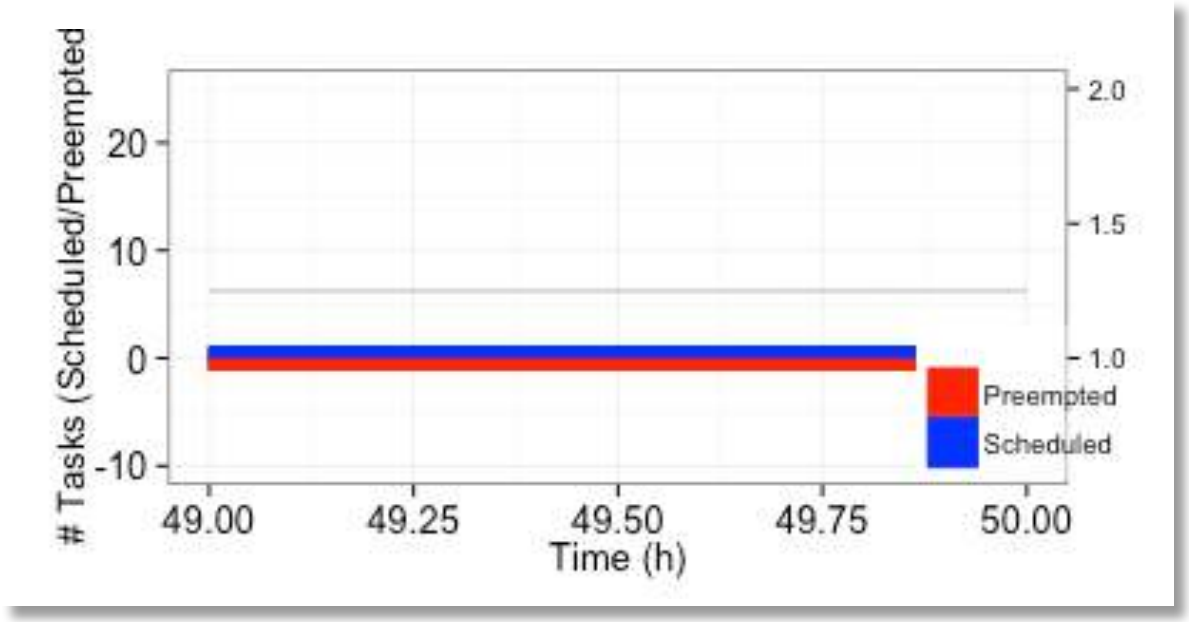
PROPORTIONAL INTEGRAL DERIVATIVE



EXPERIMENTS: MEMORY USAGE



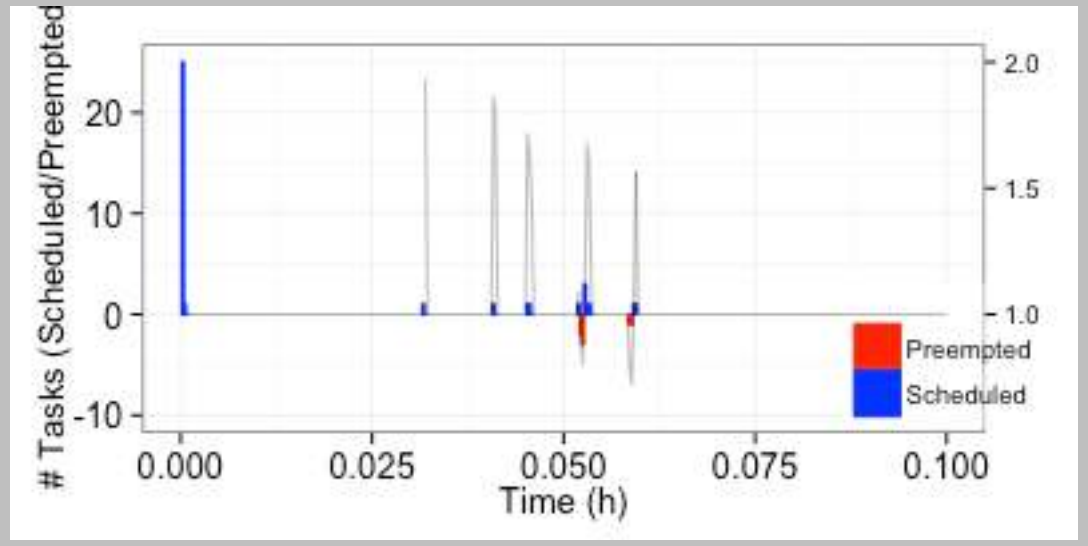
*only a few tasks (on average **less than 5**) are preempted due to **memory overflow***

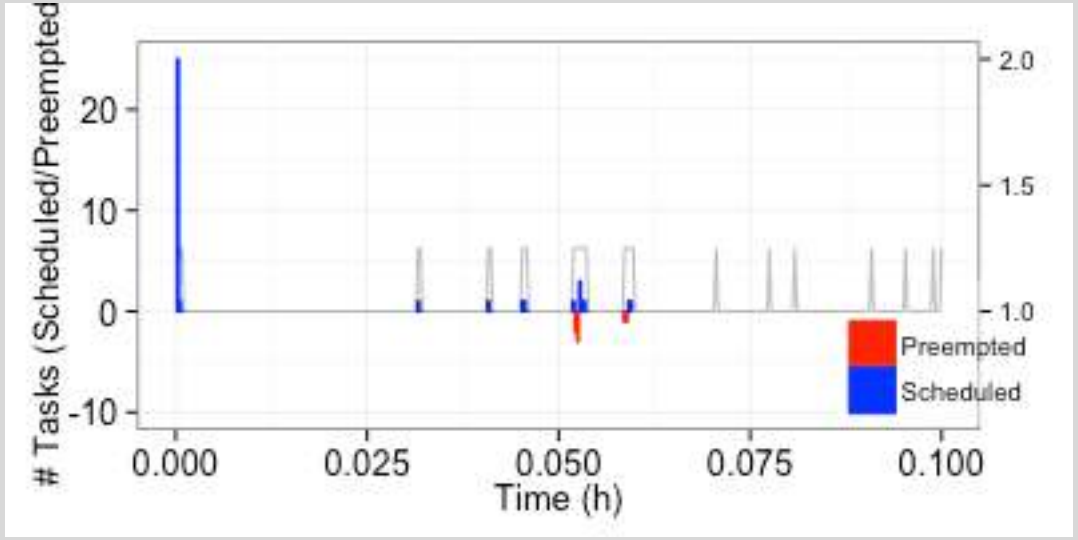


OVERALL RESULTS

Controller	# Tasks Preempted	# Cleanup Tasks
P	7225	0
PI	168	48
PID	73	4

DATA FOOTPRINT





MEMORY USAGE

TUNING PID CONTROLLERS

Execution Environment

The goal of tuning a PID loop is to make it stable, responsive, and to minimize overshooting

Ziegler-Nichols Method

1. Turn the PID controller into a P controller by setting $K_i = K_d = 0$. Initially, K_p is also set to zero

2. Increase K_p until there are sustained oscillations in the signal. This K_p value is the ultimate gain, $\boldsymbol{K_u}$

3. Measure the ultimate (or critical) period $\boldsymbol{T_u}$ of the sustained oscillations

Control Type	K_p	K_i	K_d
P	$0.50 \cdot K_u$	–	–
PI	$0.45 \cdot K_u$	$1.2 \cdot K_p/T_u$	–
PID	$0.60 \cdot K_u$	$2 \cdot K_p/T_u$	$K_p \cdot T_u/8$

Ziegler-Nichols tuning, using the oscillation method

Controller	K_u	T_u	K_p	K_i	K_d
Data Footprint	0.58	3.18	0.35	0.22	0.14
Memory Usage	0.53	12.8	0.32	0.05	0.51

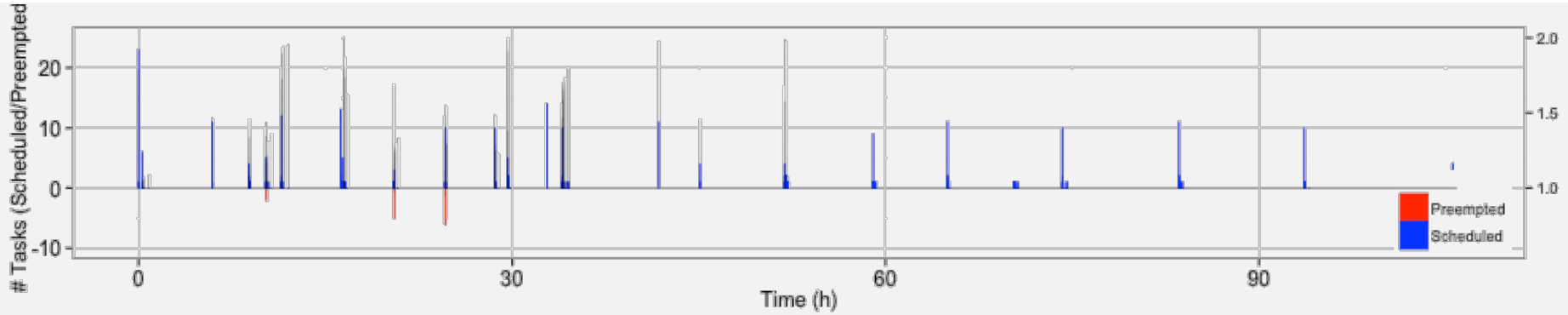
Tuned gain parameters

TUNED GAIN PARAMETERS

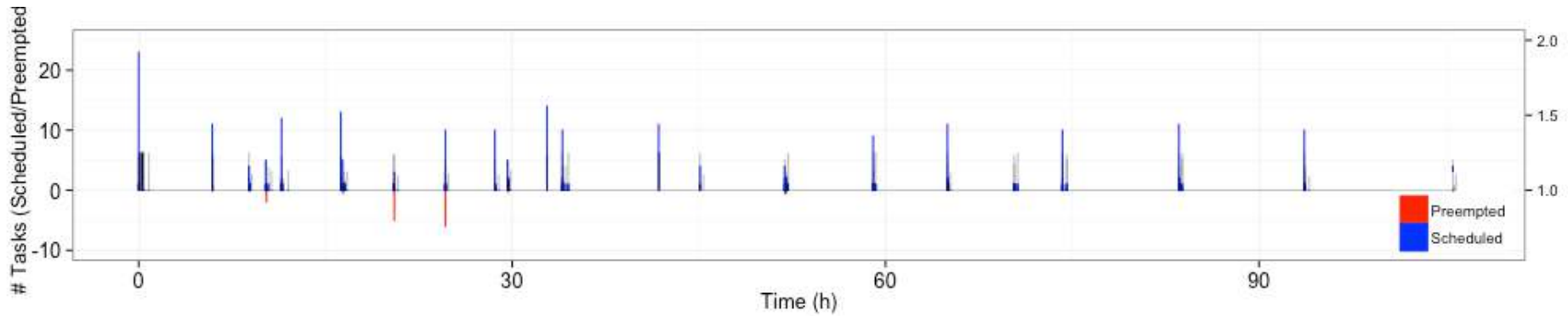
The key factor of its success is due to the specialization of the controllers to a single application

Avg. Makespan: ***107.37h***
Avg. Slowdown: ***1.01***
Preempted Tasks: ***18***
Cleanup Tasks: ***1***

DATA FOOTPRINT



MEMORY USAGE



SUMMARY

Summary

Conclusion
Future Research Directions

Conclusions

Experimental results show that faults are **detected** and **prevented** before their occur, leading workflow execution to its completion with acceptable performance

*PID controllers should be used **sparingly**, and metrics (and actions) should be defined in a way that they do not lead the system to an **inconsistent state***

Future Research Directions

We will investigate the **simultaneous** use of multiple control loops at the application and infrastructure levels, to determine to which extent this approach may negatively impact the system

USING SIMPLE PID CONTROLLERS TO PREVENT AND MITIGATE FAULTS IN SCIENTIFIC WORKFLOWS

Thank You

Questions?

Rafael Ferreira da Silva, Ph.D.

Research Assistant Professor
Department of Computer Science
University of Southern California

rafsilva@isi.edu – http://rafaelsilva.com