Design and Evaluation of an Autonomic Workflow Engine

Thomas Heinis, Cesare Pautasso, Gustavo Alonso

{heinist, pautasso, alonso}@inf.ethz.ch

ETH Zurich

Motivation

- Workflow management systems are applied to domains having unpredictable workloads

  - Scientific Workflows running in a grid environment

- Replicating functionality of the engine to achieve scalability

- But how to configure a distributed engine?

- We applied autonomic principles to a distributed workflow engine to tackle these problems
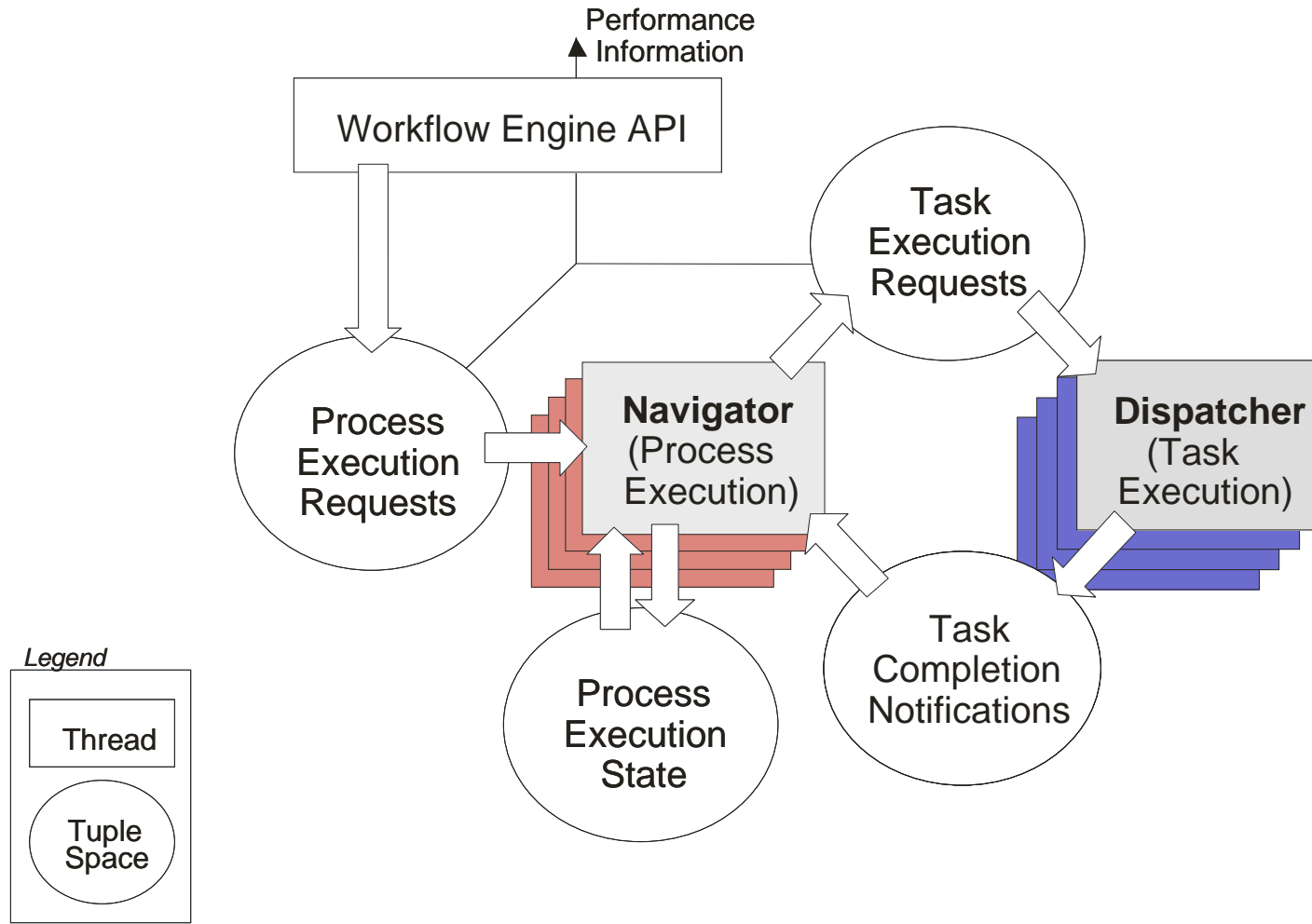
JOpera Workflow Execution Engine

- Engine that can be replicated across a cluster

- It is able to handle large number of concurrent workflow executions

- Exposes system information

- Provides means that allow reconfiguration at runtime
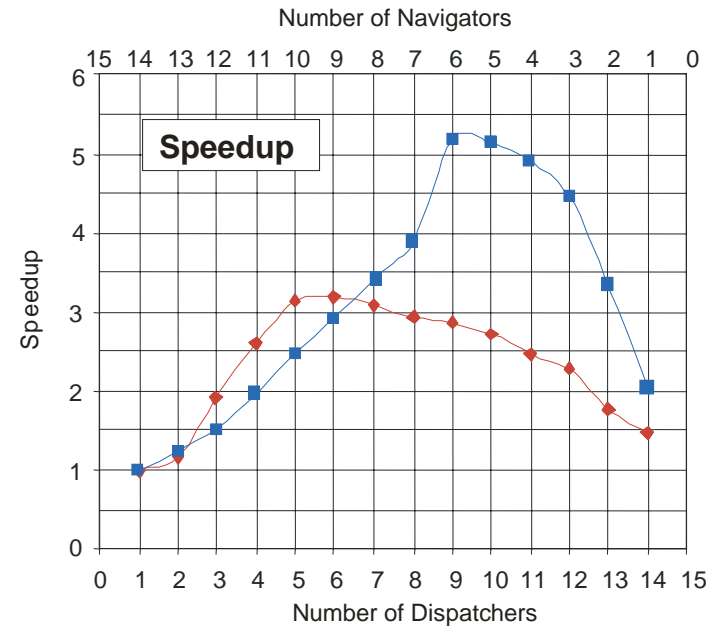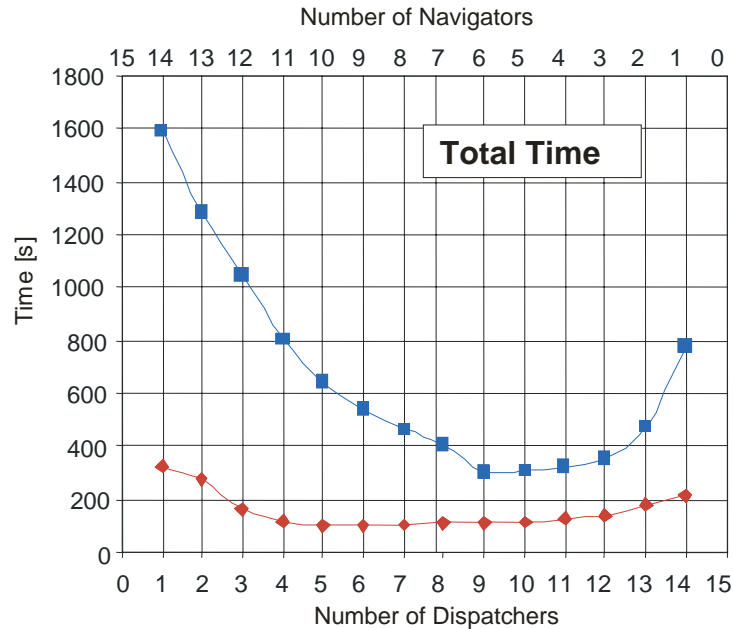
JOpera Workflow Model
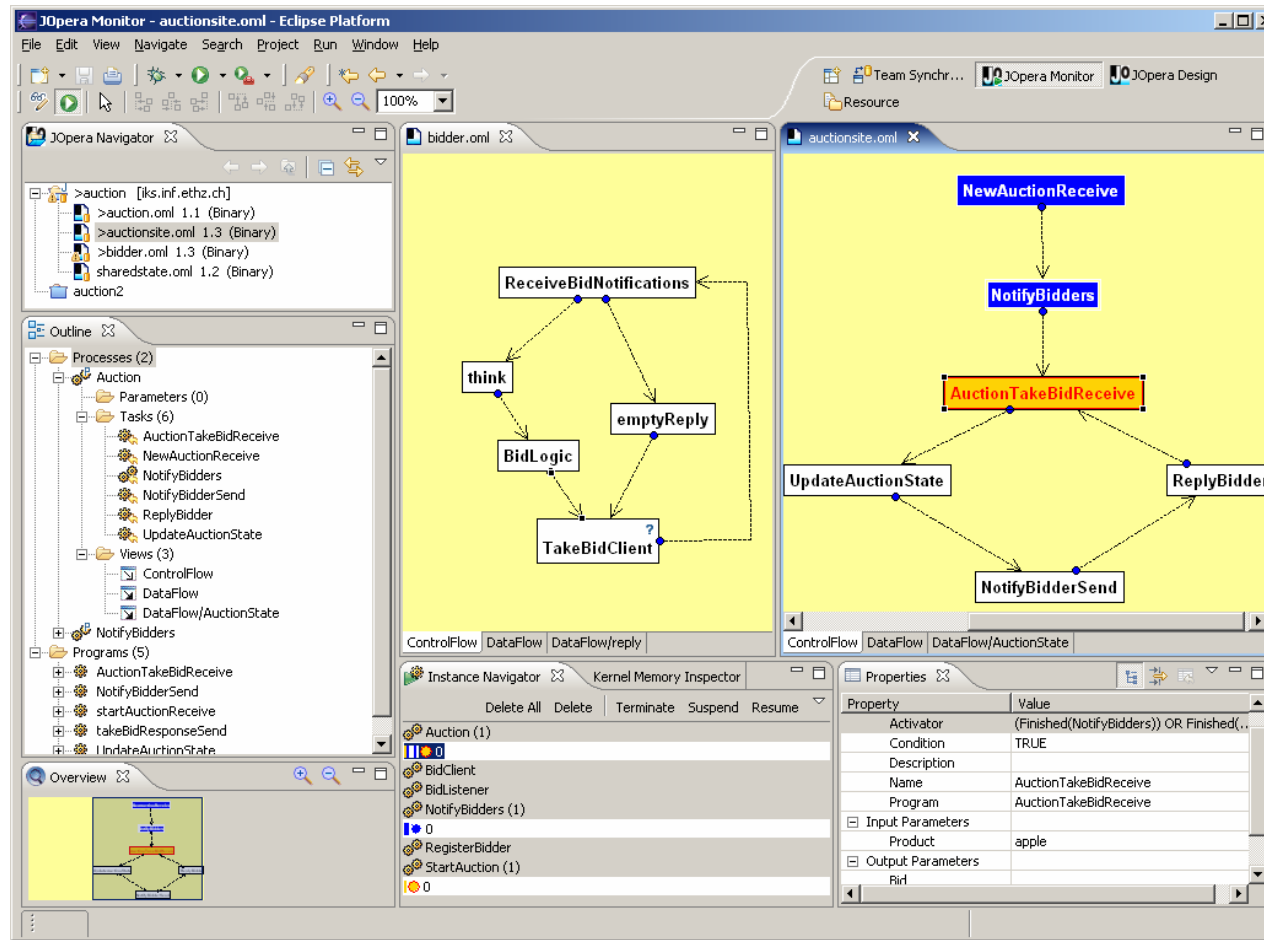
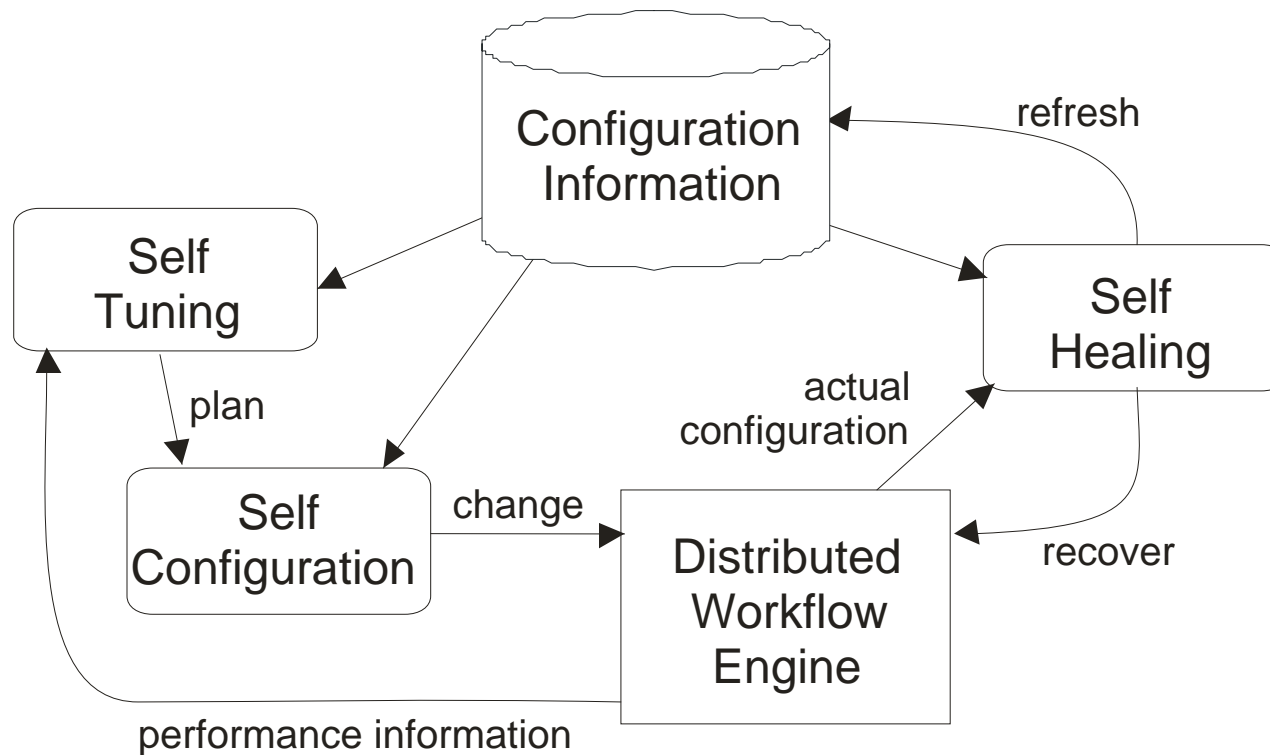# JOpera Execution Engine Architecture

Static Configuration

- ## Two different workloads:

  - Workload 0: 1000 processes with 10 parallel tasks of 0 seconds

  - Workload 20: 1000 processes with 10 parallel tasks of 20 seconds

Autonomic Controller

Self-Tuning

- **Information Strategy**
    - What performance indicators should be monitored to determine a new configuration?
        - Growth of the task execution request space (waiting tasks space) size as well as the process execution request space (waiting processes space) size
        - Current configuration information

- **Optimization Strategy**
    - What is the best configuration to balance the number of dispatcher and navigator threads?
        - The configuration is adapted such that growth of either space is bounded
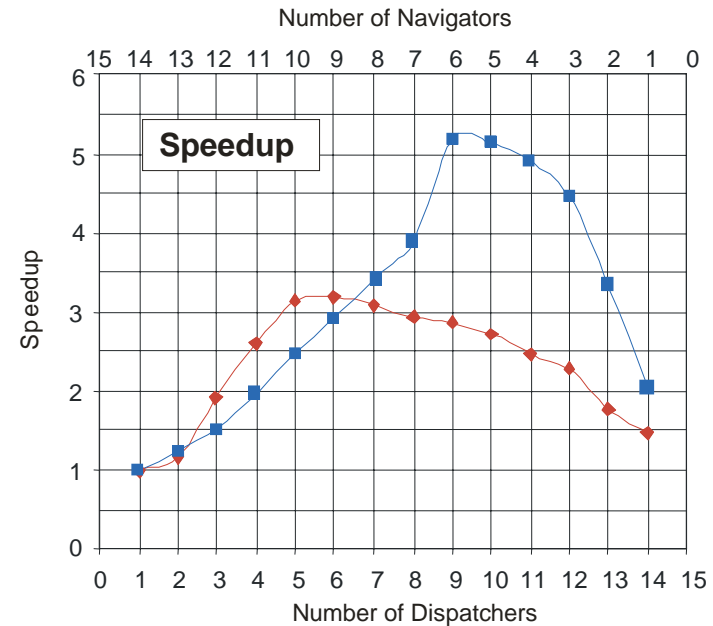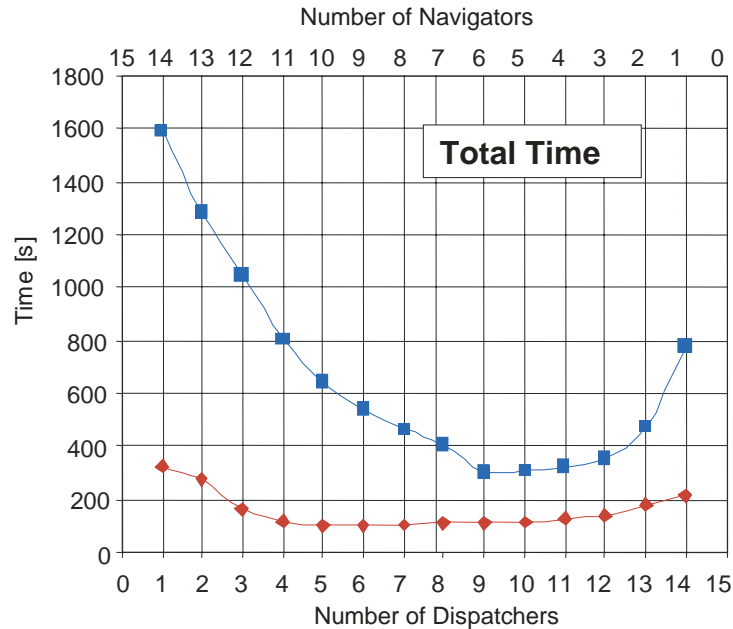
Self-Configuration Actions

- Starting Threads

- Stopping Navigator Threads

- Stopping Dispatcher Threads
  - Involves stopping the execution of tasks
    - Kill method: stop all tasks immediately
    - Stop method: let all tasks finish but do not start new tasks
  - Stopping a dispatcher may thus introduce reconfiguration overhead
    - Random selection selects dispatcher threads arbitrarily
    - Smart selection selects dispatcher that introduces the least reconfiguration overhead

Self-Healing

- Failures are detected by comparing the current configuration of the execution engine with the configuration information

- Handling dispatcher thread failures
  - State of the execution of the process is queried to determine which tasks need to be re-executed
  - Requires tasks running on the failed dispatcher to be rescheduled

- Handling navigator thread failures
  - Execution state is still available in the global process execution state space
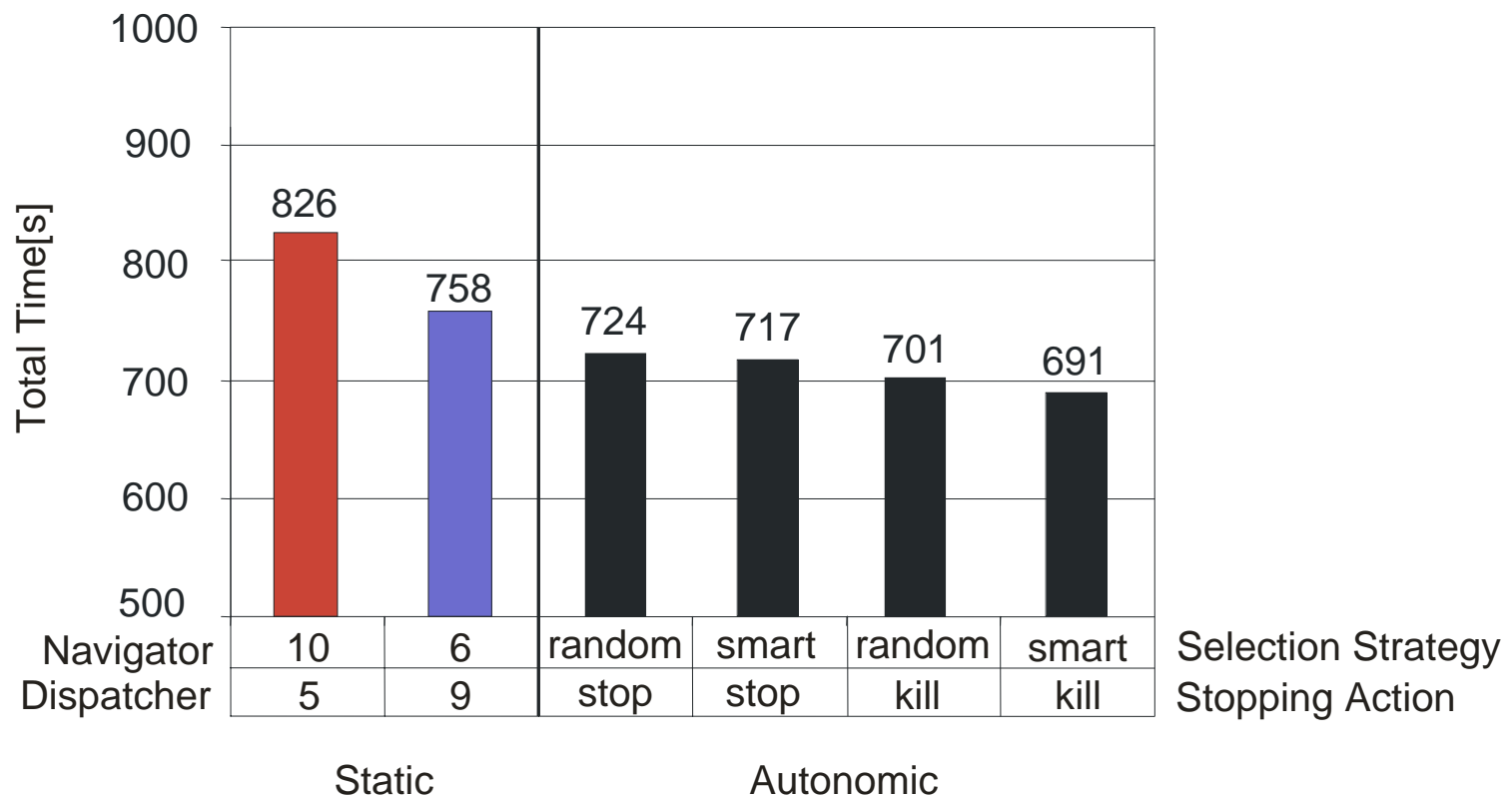
Evaluation

- ## Two different workloads:

  - Workload 0: 1000 processes with 10 parallel tasks of 0 seconds

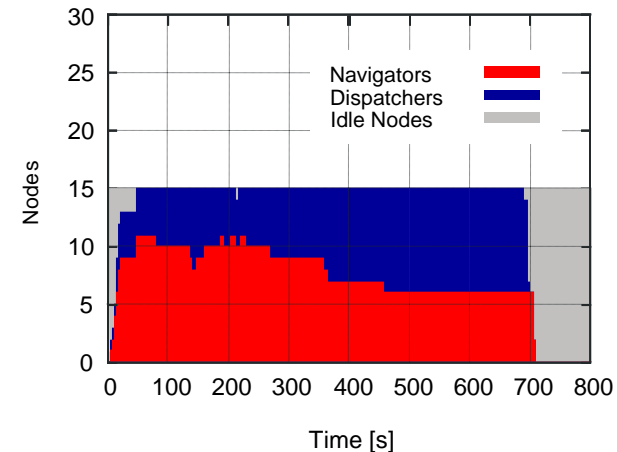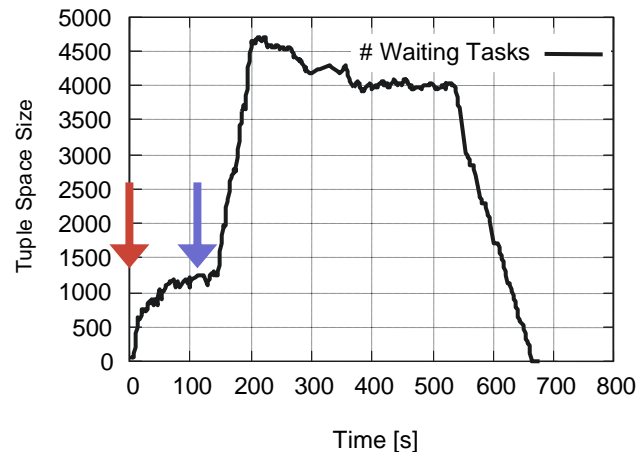 - Workload 20: 1000 processes with 10 parallel tasks of 20 seconds

Evaluation: Comparison

- Combined Workload: Workload 0, Workload 20

	Static		Autonomic				
Total Time[s]	826	758	724	717	701	691	
Navigator	10	6	random	smart	random	smart	Selection Strategy
Dispatcher	5	9	stop	stop	kill	kill	Stopping Action

Evaluation: Trace

- Combined Workload: Workload 0, Workload 20
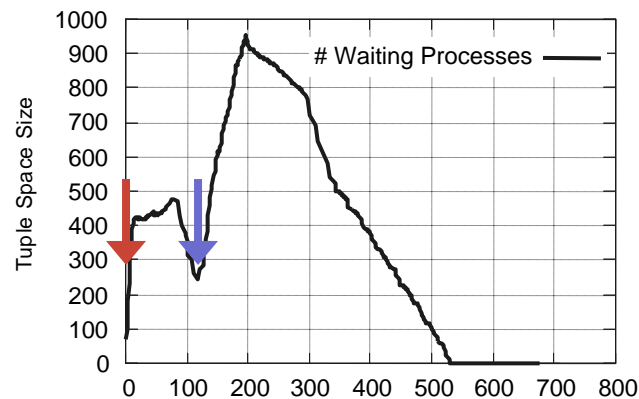
Conclusions

- Manual configuration of distributed workflow execution systems is difficult:

    - Workload characteristics are not known a priori

    - Misconfiguration leads to performance penalty

- We applied the autonomic computing paradigm on JOpera to add self-tuning, self-configuring and self-healing capabilities

- Our experiments indicate adaptation of the configuration of JOpera to a unforeseeable, changing workload

Design and Evaluation of an Autonomic Workflow Engine

Thomas Heinis, Cesare Pautasso, Gustavo Alonso
{heinist, pautasso, alonso}@inf.ethz.ch
ETH Zurich

available at http://www.jopera.org