TOWARDS A UNIFIED TELEMETRY SERVICE FRAMEWORK FOR HPC ENVIRONMENTS

Ole Weidner
School of Informatics
University of Edinburgh
ole.weidner@ed.ac.uk

Adam Barker
School of Computer Science
University of St Andrews
adam.barker@st-andrews.ac.uk

Malcolm Atkinson
School of Informatics
University of Edinburgh
malcolm.atkinson@ed.ac.uk

INTERNATIONAL WORKSHOP ON RUNTIME AND OPERATING SYSTEMS FOR SUPERCOMPUTERS

WASHINGTON, D.C., USA, JUNE 27, 2017

OUTLINE

1. Application Challenges and Motivation
2. Telemetry as HPC Platform Service
3. Context Graph Model
4. Interaction and Interface
5. Prototype
6. Discussion

DEFINITION

HPC Telemetry Data

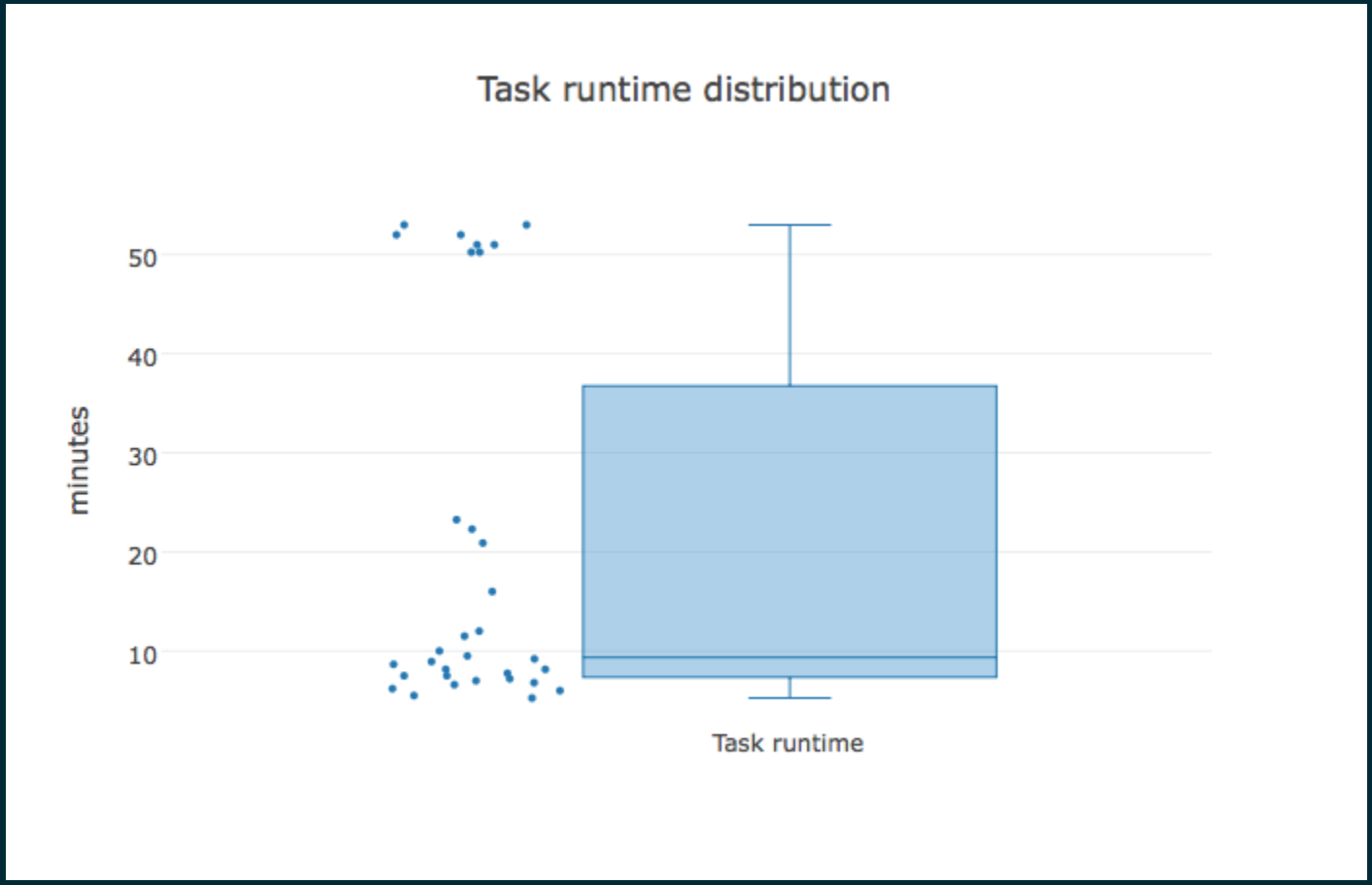
Any data that describes the state of an HPC platform and the state of the process-based representation of the applications running on it.

1

APPLICATION CHALLENGES & MOTIVATION

A NORMAL DAY AT THE OFFICE

Strange runtime distribution of homogeneous tasks

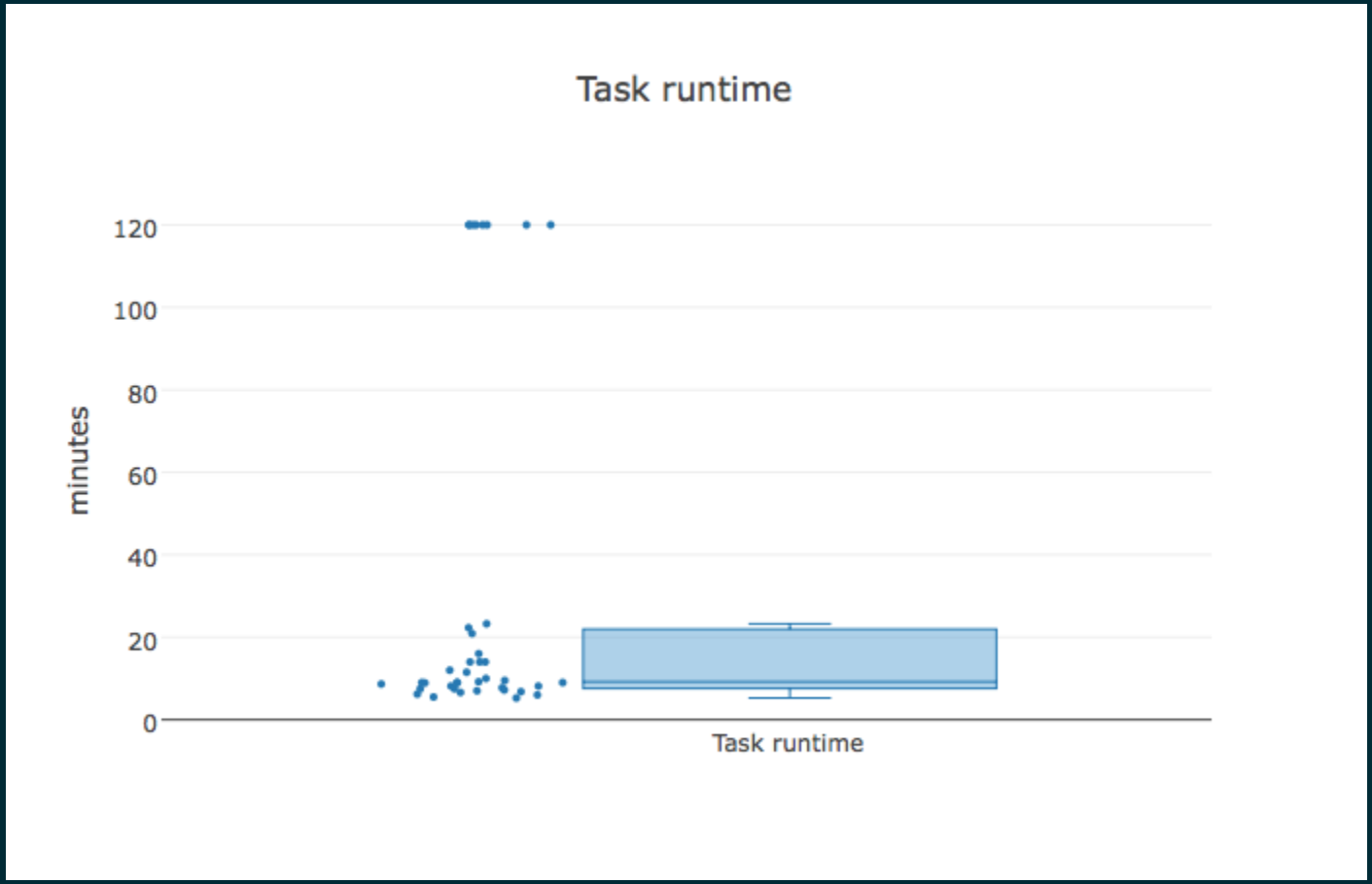


FINDING THE CULPRINT

- Added logging to the application to understand where time is spent
  - Some tasks spent 10x longer downloading input dataset
  - A faulty edge switch caused external connectivity issues on some nodes
- Introduced helper tasks that collect process-level metrics
  - Some tasks spent a hughe amount of time in IO Wait
  - A strange problem with Lustre caused slow filesystem I/O on a small set of nodes

ANOTHER INTRESTING CASE

Again, an unexpected runtime distribution of supposedly homogeneous simulation tasks



FINDING THE CULPRINT

- Used the same instrumentation strategy
  - Outlier tasks run out of memory and stall
  - Specific structural properties of the input data would cause the algorithm to take a different trajectory

CONSEQUENCES

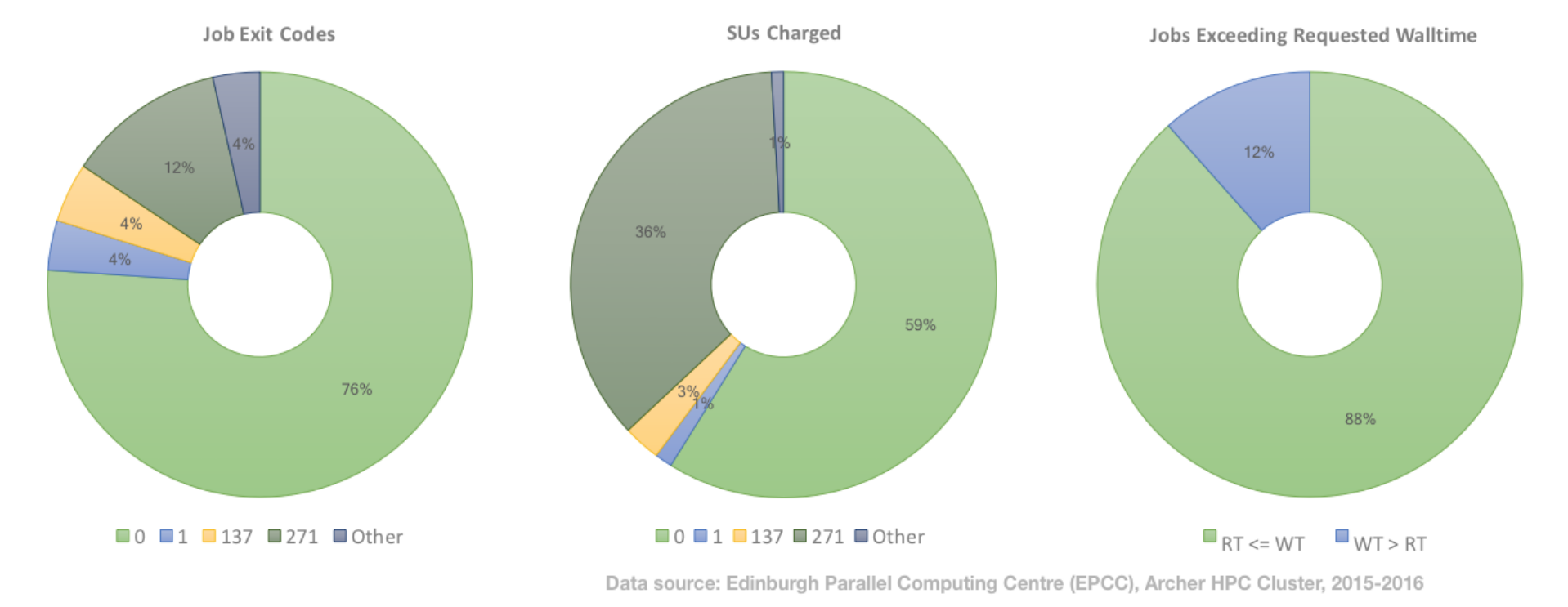
- We encountered unexpected "dynamic behavior", both on the system as well as on the application side
- Knowing that these are no edge cases, we started making our "debugging" approach a more vital part of the application framework:
  - Collecting process- and OS-level information during all runs
- Applying simple adaptive strategies to mitigate issues at runtime:
  - Blacklist 'weird' nodes
  - Reducing the task-packing (preempt other tasks on the node) when memory usage exceeds threshold

EXPERIENCE & LESSONS LEARNED

- Instrumetation requires a lot of effort
- Collecting and analysing data (at scale) is non-trivial
- Interpreting and feeding the data to the application is difficult
- Existing tooling is sparse and mostly geard toward post-mortem, parallel code debugging
- Without knowing and understanding the platform "anatomy" and context, data can be difficult to interpret, e.g., what is considered "poor" I/O, what is the spatial layout of processes across nodes?

EXPERIENCE & LESSONS LEARNED *CONT.*

- Application-specific instrumentation is wide spread technique to mitigate heterogeneity, dynamic behavior, etc.
- Adressing the issue is expensive, but ignoring it can be expensive, too:

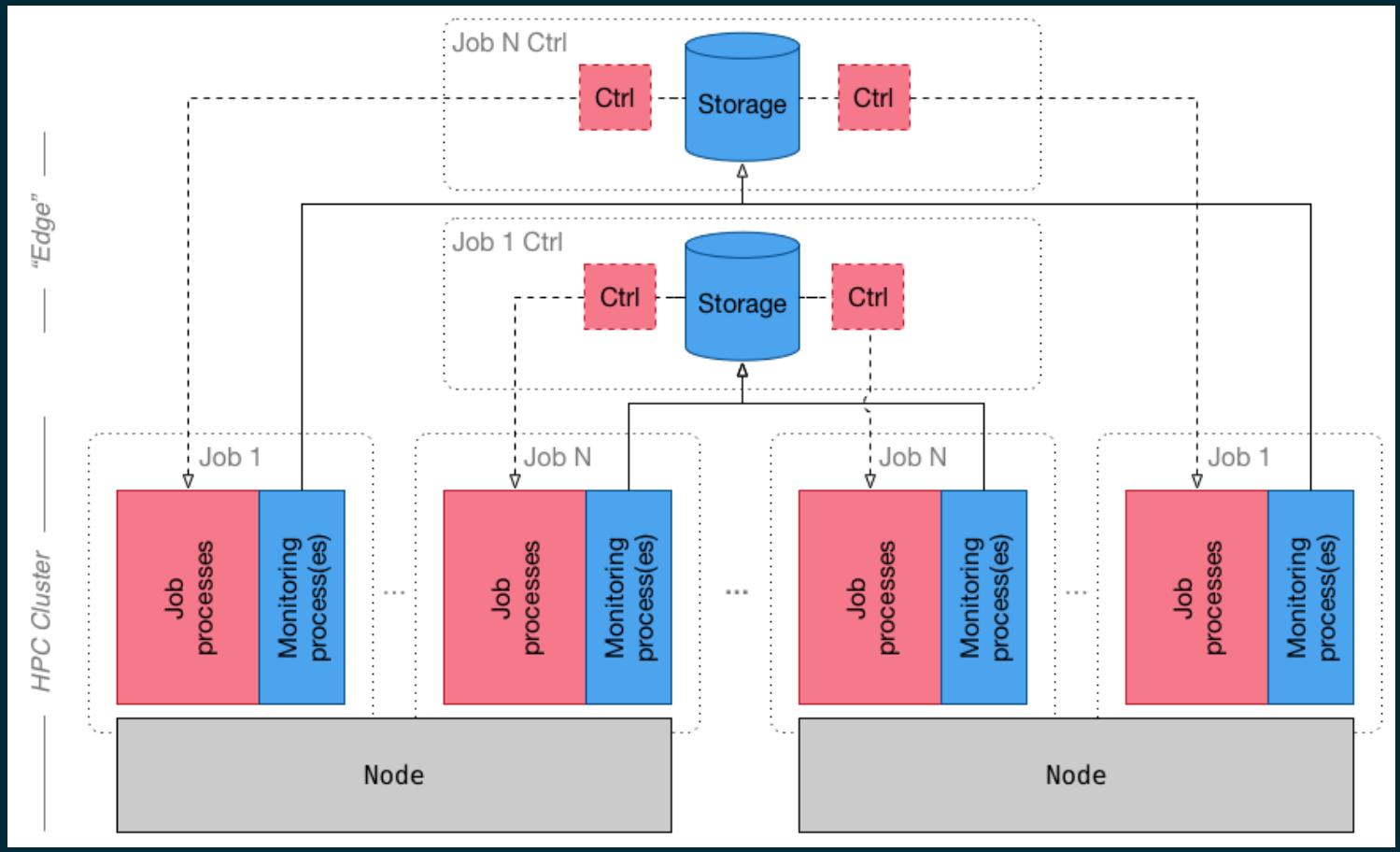


2

TELEMETRY AS HPC PLATFORM SERVICE

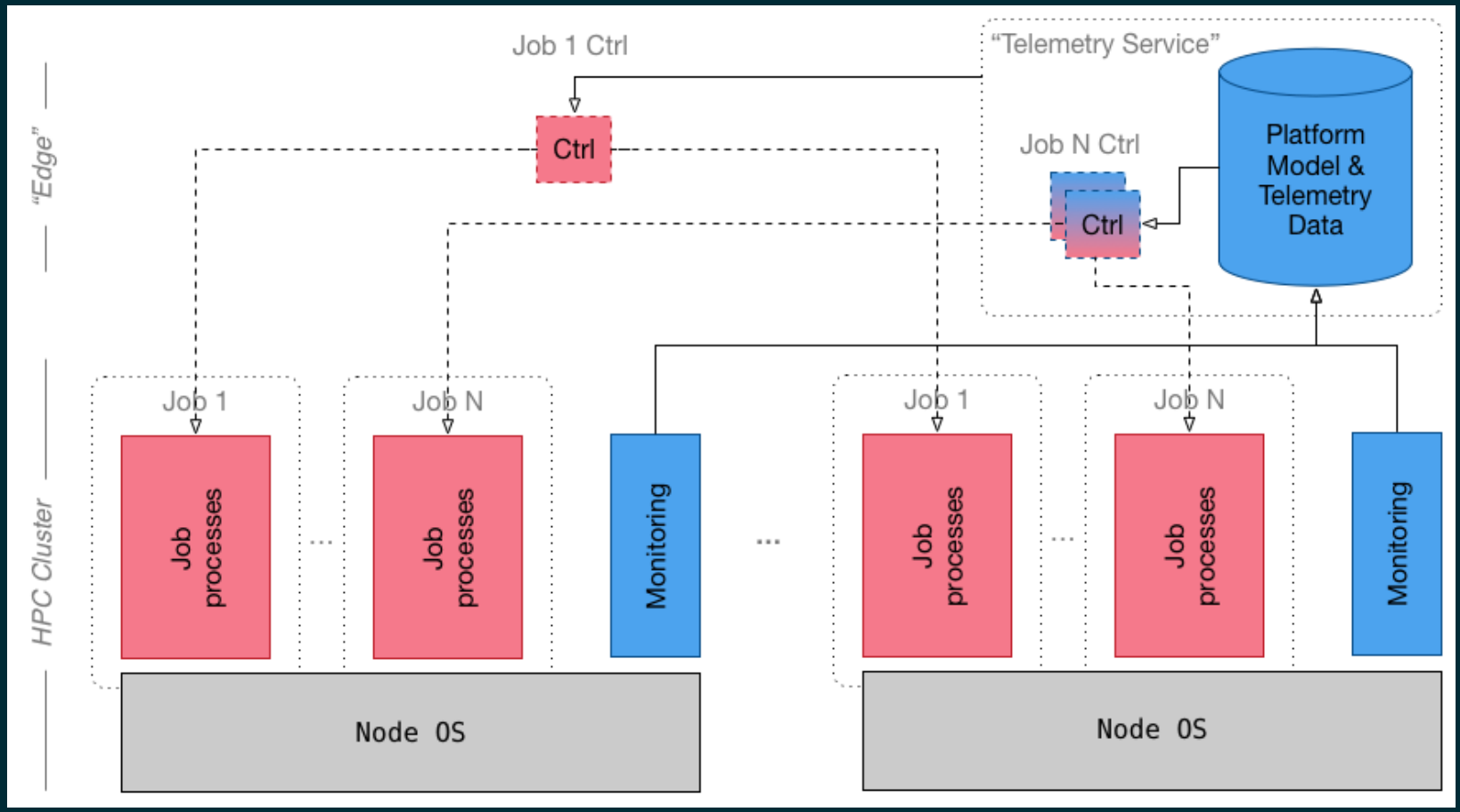
STATUS QUO: APPLICATION-DRIVEN

Application-level collection and processing of telemtry data can cause a lot of overhead.



PLATFORM SERVICE APPROACH

Telemetry service takes over data collection and provides data access and higher-level functions to applications



REQUIREMENTS

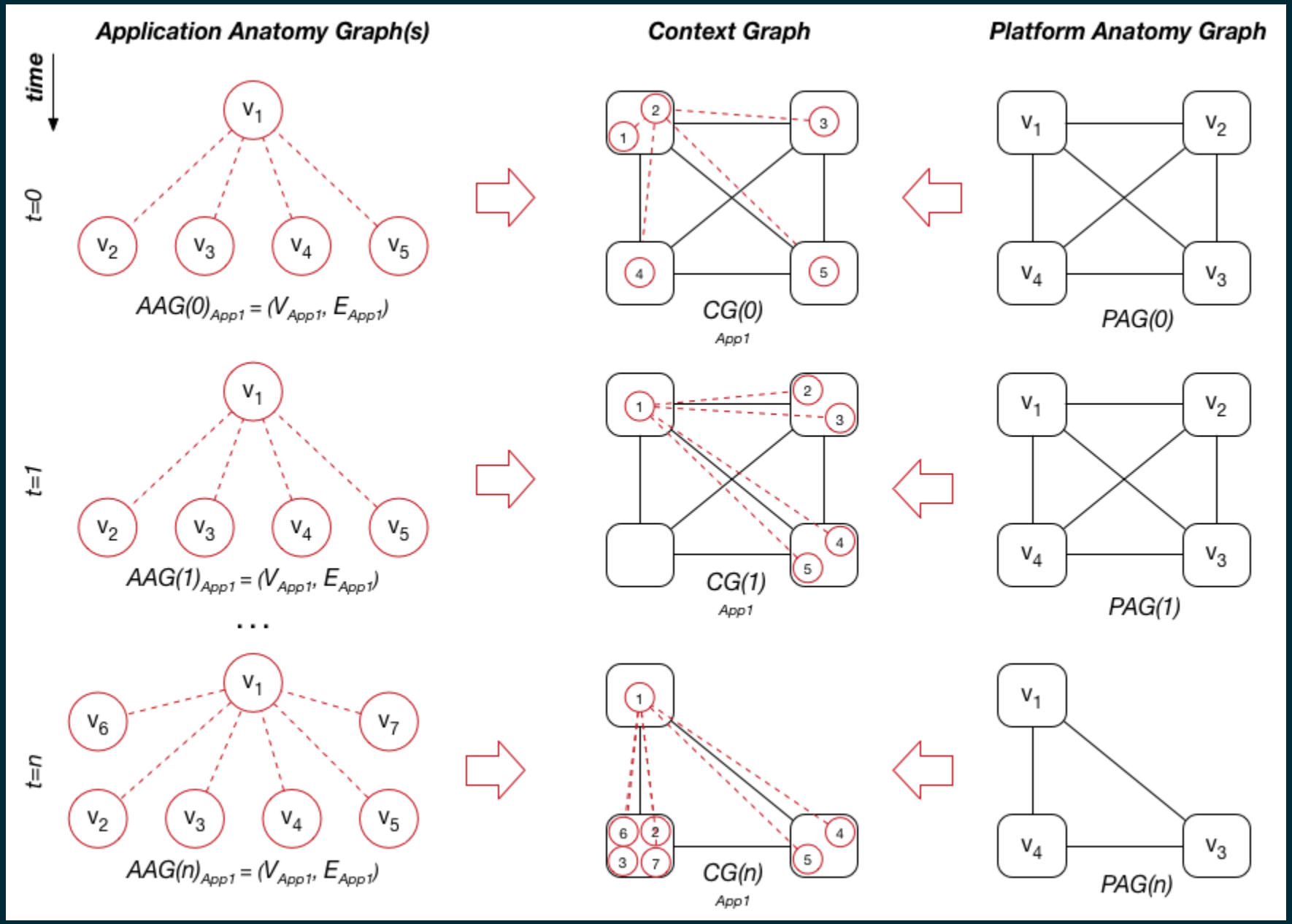
- Captures the time-variant physicla anatomy and properties of applications
- Captures the time-variant anatomy and properties of the HPC platform
- Describes the mapping between the two (contex!)
- Allows for arbitrary levels of detail
- Provides programmatic access to the data
- Allows offloading data analytics, e.g. extracting trends from streams of raw data
- Has notifications capabilities

REQUIREMENTS *CONT.*

- Keeps historic data (possibly in condensed form)
- Is deployable at scale (think exascale!)
- Consistent across platforms

3

CONTEXT GRAPH MODEL


Application Anatomy Graph(s)
Context Graph
Platform Anatomy Graph
time
t=0
t=1
t=n
v1
v2
v3
v4
v5
v6
v7
AAG(0)App1 = (VApp1, EApp1)
AAG(1)App1 = (VApp1, EApp1)
AAG(n)App1 = (VApp1, EApp1)
CG(0)
App1
CG(1)
App1
CG(n)
App1
PAG(0)
PAG(1)
PAG(n)

GRAPH-BASED MODEL

- Provides the context in which time-series can be embedded
- We use attributed graphs to describe entities and their relationships
- Graphs provide a intuitive way to model arbitrary levels of complexity
- A single **context graph** (*CG*) captures the connections between the **platform anatomy** (sub-)graph (*PAG*) and the **application anatomy** (sub-)graphs (*AAG*)

SPATIAL-TEMPORAL DYNAMICS

- Anatomy and structure of platform and applications is not static:
  - Application process start and stop
  - Nodes appear and disappear
  - Hardware (e.g., GPUs or FPGAs) is added
  - ...
- All nodes and edges have timestamps that qualify their existence
- To get a snapshot of the platform and applications at a specific point in time, the graph can be queried for a specific time or time range

4

INTERACTION AND INTERFACE

USER- / APPLICATION-FACING API

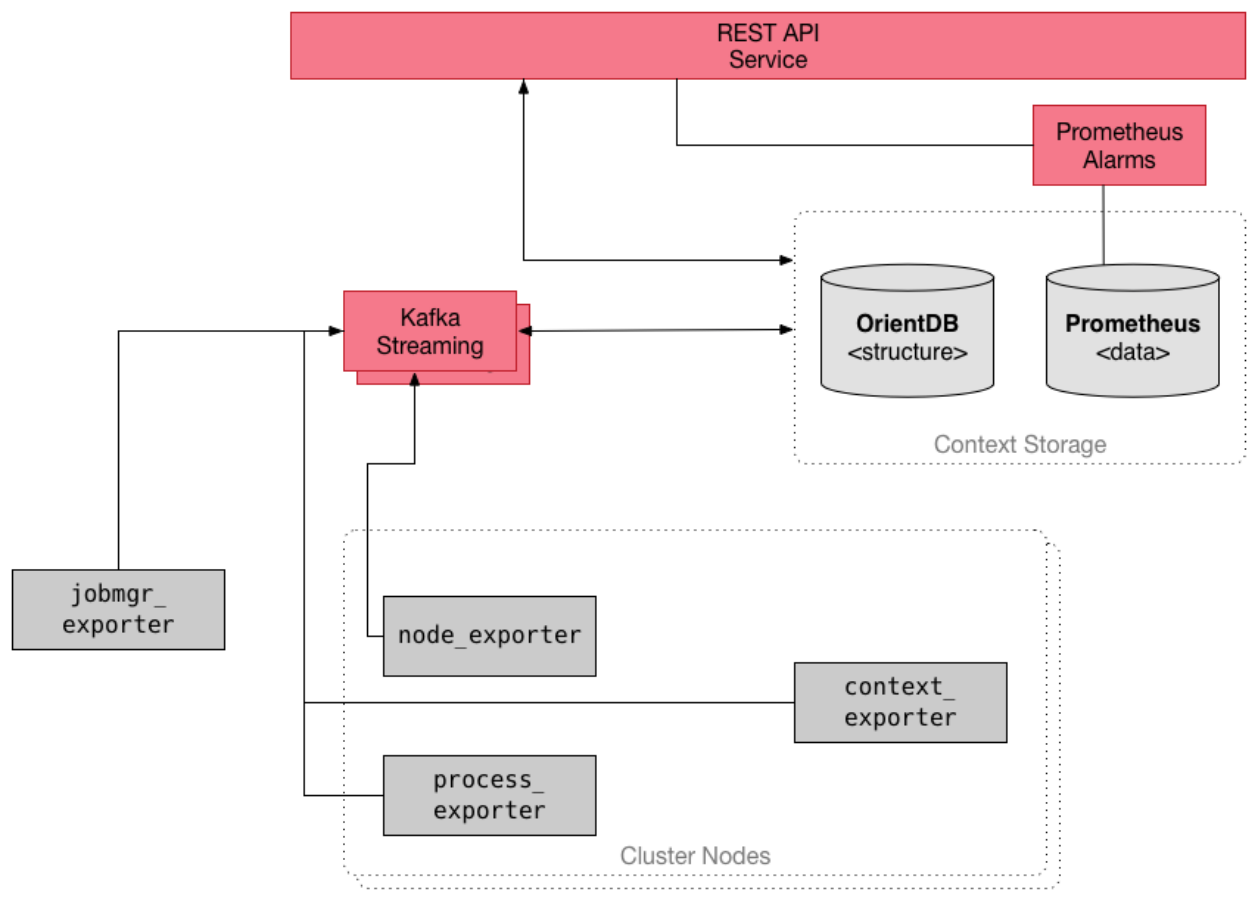
- Language-Agnostic HTTP/REST API allows to:
  - Explore / traverse the context graph
  - Register simple "server-side" "derived metrics" functions
  - Define and register call-backs (Websockets)
  - GraphQL for complex graph queries

```
{
 process(id: 1) {
   siblings {
     processes {
       cpu_iowait
       memory_uses
     }
   }
 }
}
```

5

PROTOTYPE

SYSTEM COMPONENTS



6

DISCUSSION

This is how we envision an ideal system from the application developer's / user's perspective

THANK YOU

Slides available online:
https://oweidner.github.io/ross-2017-talk