User Centric Monitoring – a redesign and novel approach in the STAR experiment

D. Arkhipkin

J. Lauret

J. Zoulkarneeva

BROOKHAVEN
NATIONAL LABORATORY

Outline

User-Centric Monitoring – Grid/Cloud Logger and Complex Event Processing

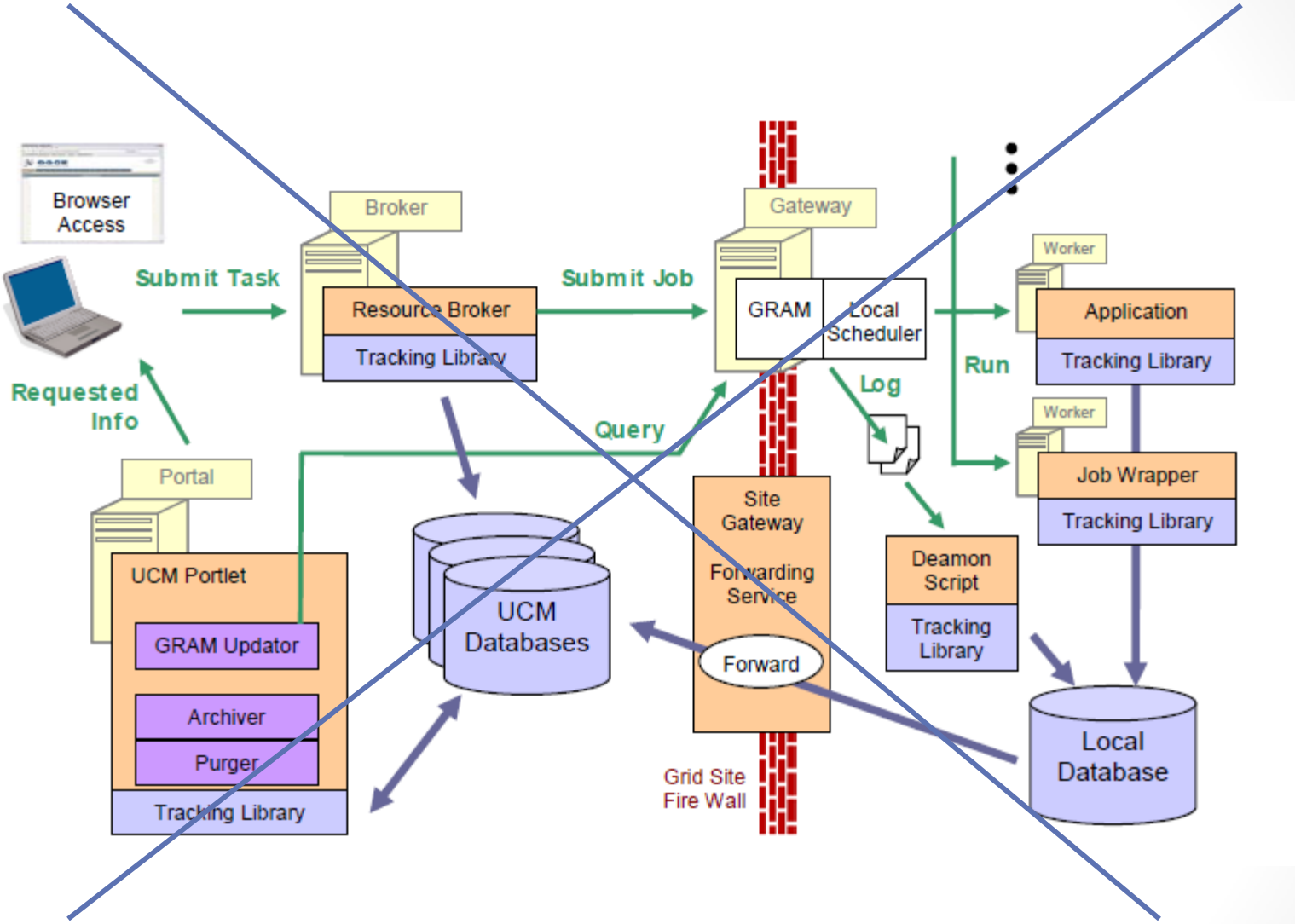
1. Application Message Logger Framework for distributed job processing (Grid & Cloud)
2. Complex Event processing – proof of principle in online MetaData filtering context
3. Summary and outlook

APPLICATION MESSAGE LOGGER FRAMEWORK FOR DISTRIBUTED JOB PROCESSING (GRID & CLOUD)

User Centric Monitoring – UCM

- **User-centric monitoring idea:** serve a rich set of intuitive job and application monitoring information to scientists so that they can be more productive
  - Started as an SBIR in 2007 - D.A. Alexander, C. Li, J. Lauret, V. Fine "User Centric Monitoring (UCM) Information Service for the Next Generation of Grid-enabled Scientists", CHEP 2007 proceedings, *J. Phys. Conf. Ser.* **119** 052001 doi:10.1088/1742-6596/119/5/052001 (2008)
  - Project came short on deliverables – UCM logger (appender) handled local caching of messages + push to a DB
  - In //, STAR investigated the use of AMQP online - D Arkhipkin, J Lauret, W Betts, G Van Buren – "Online Meta-data Collection and Monitoring Framework for the STAR Experiment at RHIC", *J. Phys.: Conf. Ser.* **396** 012002 doi:10.1088/1742-6596/396/1/012002 (2012) Full use in RHIC Run 13

- "**Cloud Logger**" idea was revived …

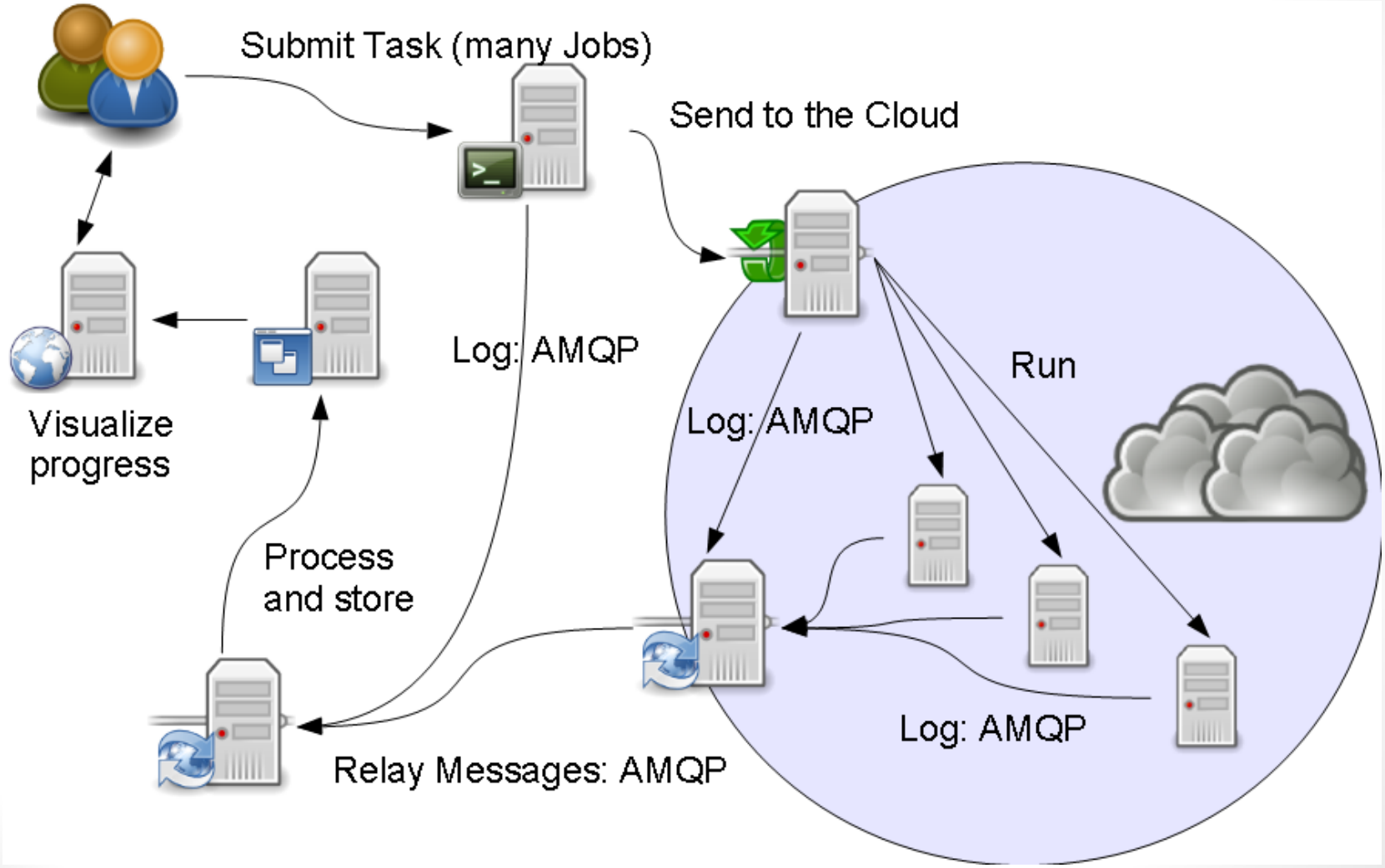

Browser Access
Submit Task
Broker
Resource Broker
Tracking Library
Submit Job
Gateway
GRAM
Local Scheduler
Worker
Application
Tracking Library
Run
Worker
Job Wrapper
Tracking Library
Log
Requested Info
Query
Portal
UCM Portlet
GRAM Updator
Archiver
Purger
Tracking Library
UCM Databases
Site Gateway
Forwarding Service
Forward
Grid Site Fire Wall
Deamon Script
Tracking Library
Local Database

Evolution to a Grid/Cloud Logger

- Implements an extended and reworked version of the User-Centric Monitoring Model
- Upgraded UCM framework (a.k.a CloudLogger)
  - set of logging interfaces
  - reliable messaging bus
  - a schema-free database backend
  - a Web interface to present logged information to users in a user-centric view rather than and administrative-centric point of view

- **Primary features:**
  - Multi-tier architecture allows scalability and easier maintenance
  - New schema-free database backend ensures smooth upgrades in a future
  - Full support of the Grid/Cloud usage and external site logging
  - Improved Ajax-based user interface, which allows to track jobs performance via automatic job statistics gathering and histogramming

Design overviews



Message hierarchy

- Leverages the UCM lesson learn : Structured properties into three categories: **user task level**, **job level**, and **job event level**
  - Directly translates to our data store collections: **Tasks**, **Jobs**, **Events**.
  - Document-based storage allows in-place updates for event messages, job status and task parameters..
  - Structures presented in JSON format – native to db backend and web services
- Important note – messages are not only "text" but key:values
  - Some key:values are used to display process information (memory consumption, CPU time, …)
  - Web front end can collect all info and display

Task:

```
{
  "task_id" : <
  "broker_tasl
  "broker_id":
  "requester_i
  "size": <num
  "site_locatio
  "submission
  "description
  "messages"
}
```

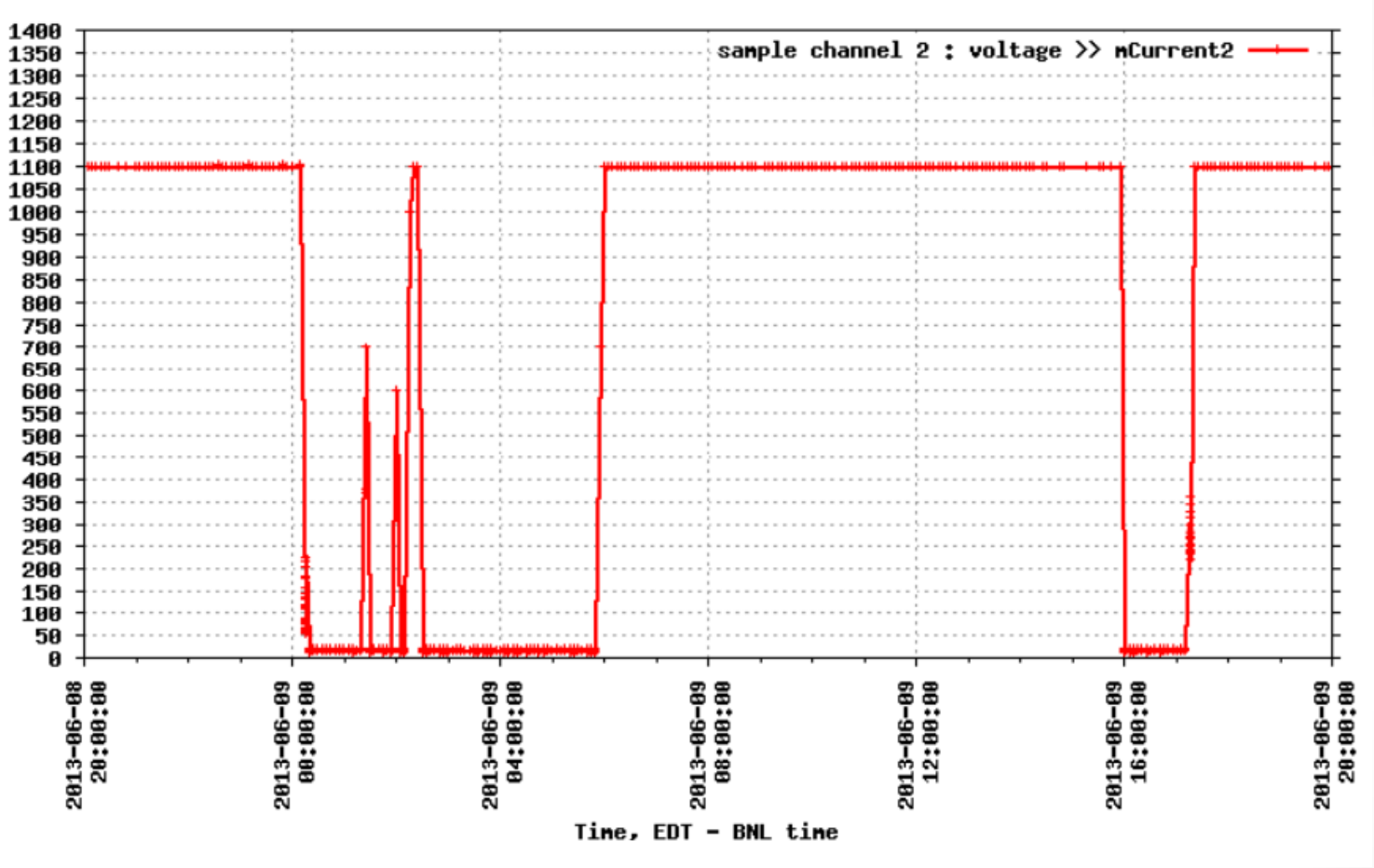
Job:

```
{
  "job_id": <assign
  "task_id": <task_
  "stage_id": <subi
  "ts_submit_local"
  "ts_submit_cloud
  "execution_node
  "messages": [ {"u
}
```

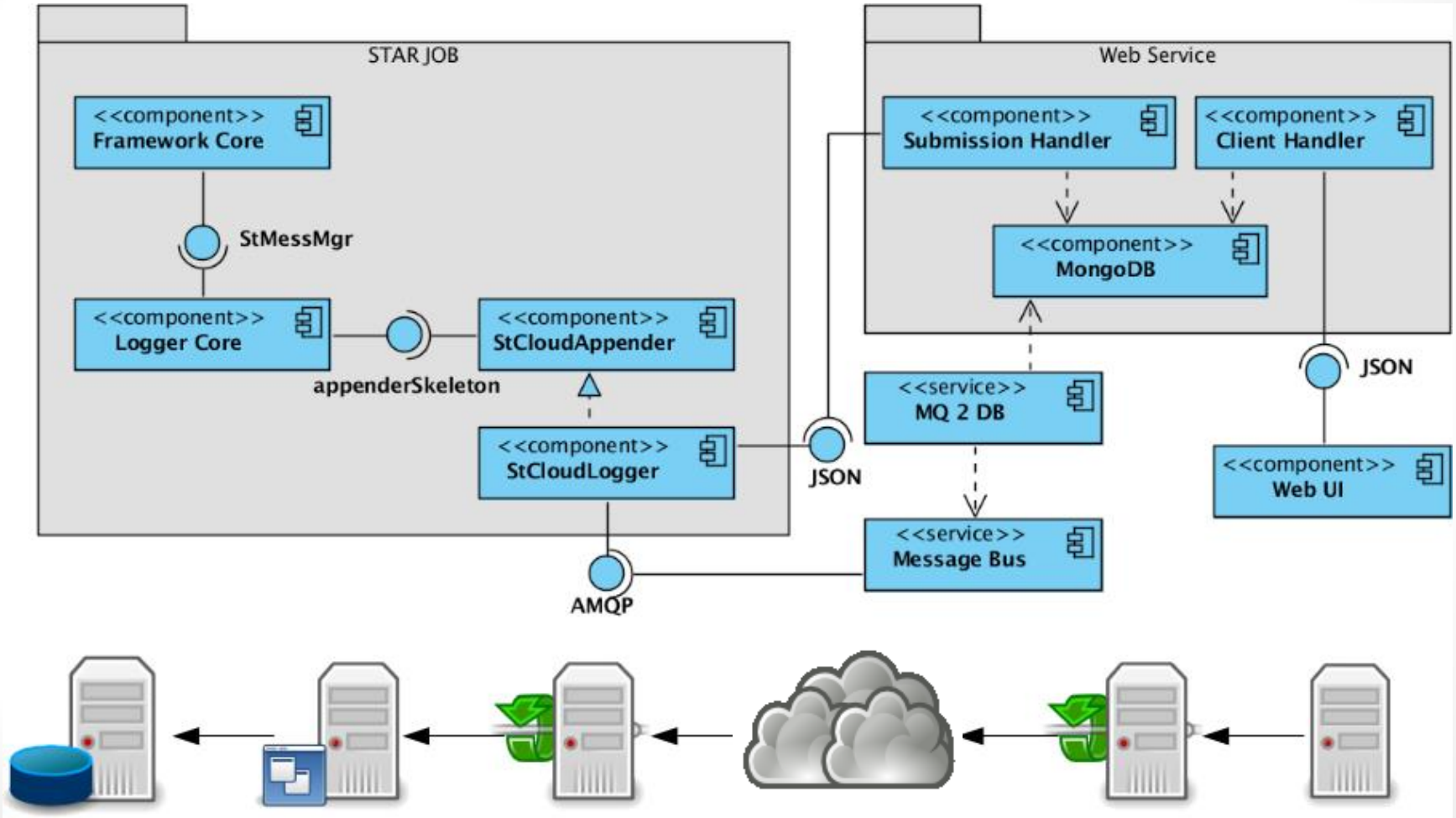
Event:

```
{
  "task_id": <>,
  "job_id": <>,
  "messages": [
      { "mtime": <timestamp>,
        "context": <val>,
        "stage_id": <id>,
        "level": <id>
         "message": {"key1": value1,.. "keyN": valueN}
      }
  ]
}
```

Allows msg="hello world" but also CPUtime=12sec, …


sample channel 2 : voltage >> mCurrent2
1400
1350
1300
1250
1200
1150
1100
1050
1000
950
900
850
800
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
2013-06-08 20:00:00
2013-06-09 00:00:00
2013-06-09 04:00:00
2013-06-09 08:00:00
2013-06-09 12:00:00
2013-06-09 16:00:00
2013-06-09 20:00:00
Time, EDT - BNL time

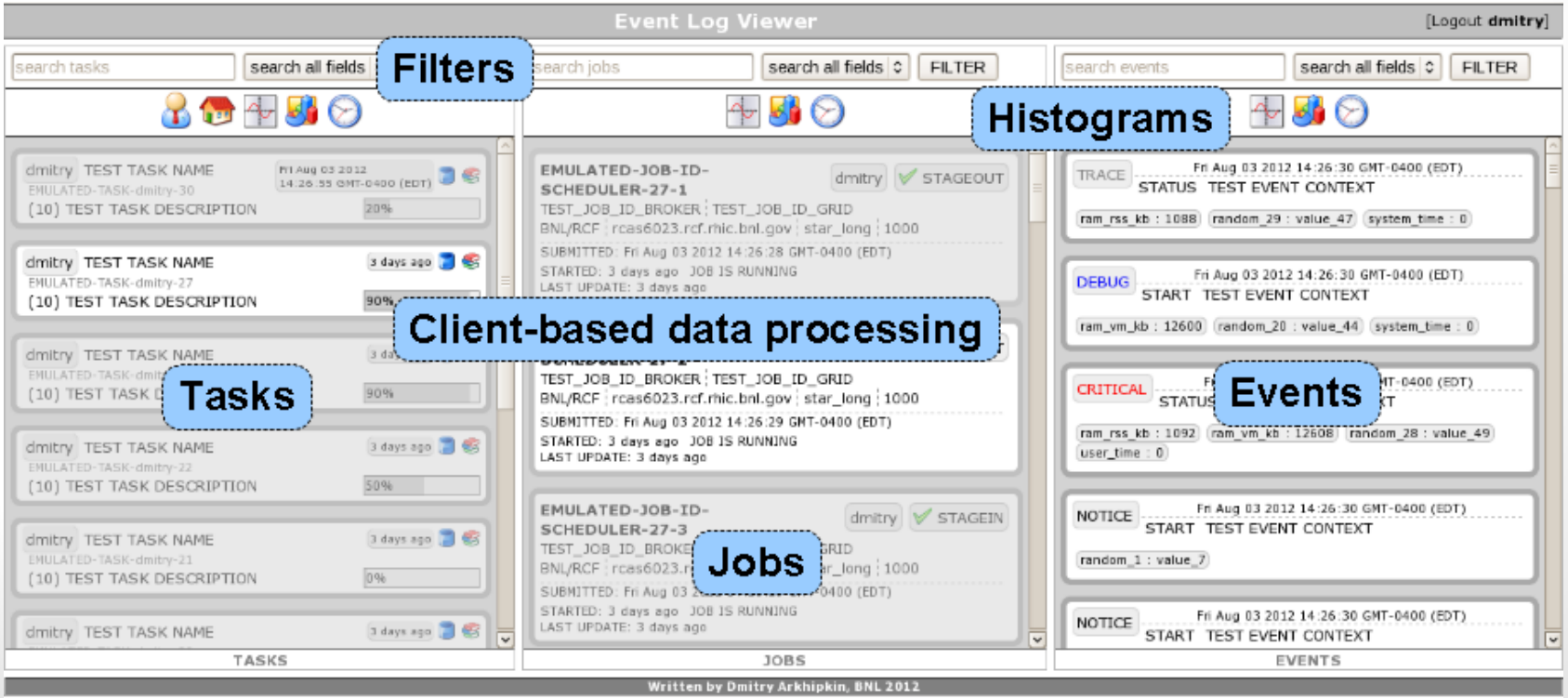
Component refactoring



May use the messaging protocol or may skip it and use http (if possible)
Apache qpid used as message broker to ensure messages reach their destination

Web front-end

- Three-column layout: tasks, jobs, events – dynamically populated upon request
- Virtually no data conversion happens, as both web client and database backend are using JSON (BSON) internally
- Features embedded search filters, automatic histogramming and more!

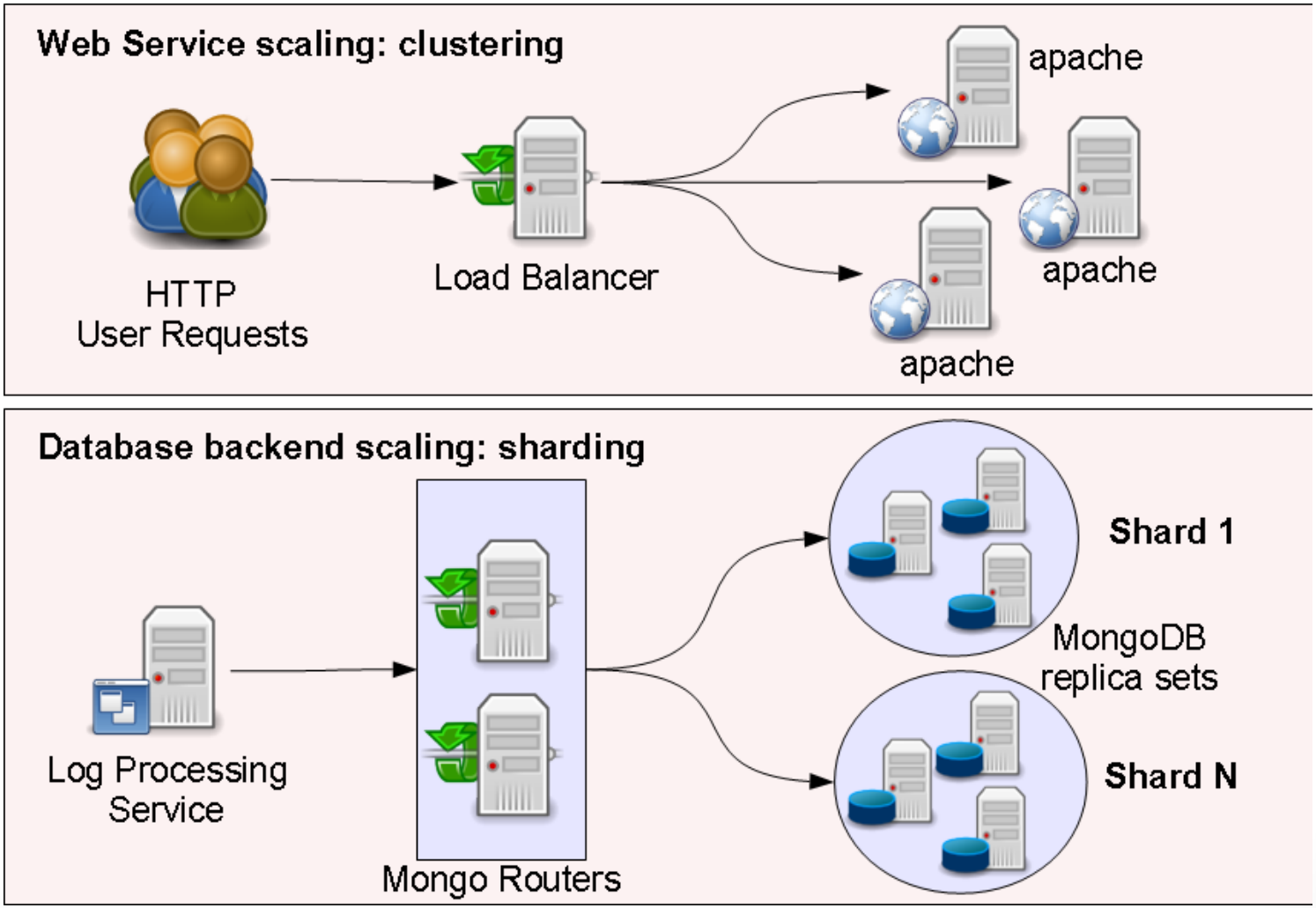


Performance and scaling

Key factors: (1) messaging broker throughput (2) web service scalability (3) database backend scalability

- **Messaging bus** performance and scaling
  - Depends on AMQP implementation.
    We use qpid daemon, supported by Red Hat, with up to a ~ million messages per second throughput on a tuned system, or ~ 200k messages/second on a typical commodity system
  - STAR requirements are fully satisfied as our demands are modest:
    For 5k jobs up to 10,000 messages/sec on average, with peaks at ~50,000 messages/sec
    Even a growth by x50 would still be fine on our tuned system
- **Web Service** scaling
  - Allows flexible scaling by adding web servers on demand, presumably utilizing load-balancing front-end router
- **Database** performance and scaling
  - Assuming in-place updates most of the time, and automatic partitioning across working nodes (sharding) of the MongoDB we do not see any constrains on the db backend side for both writes and reads

Scaling



sharding = horizontal partitioning.
Ex: db can be split by taskID (many jobs, event more messages)

COMPLEX EVENT PROCESSING – PROOF OF PRINCIPLE IN ONLINE METADATA FILTERING CONTEXT

General idea

- For any “events” (messages or online MetaData)
  - Event is created (various data sources)
  - Event is distributed to clients (pushed to MQ)
  - Event is stored (db, memcache) and/or visualized
  - Then, event(s) is/are read from database, matched to other events from the same or different db, processed and analyzed by custom scripts – Not really “integrated” … but it works …

- We could do better
  Event streams need to be processed simultaneously and be acted upon to discover event correlation, relationship and patterns

  Essentially, we needed to add multi-stream, multi-event processing capabilities to STAR message/event collection processing system

  CEP = Complex Event Processing

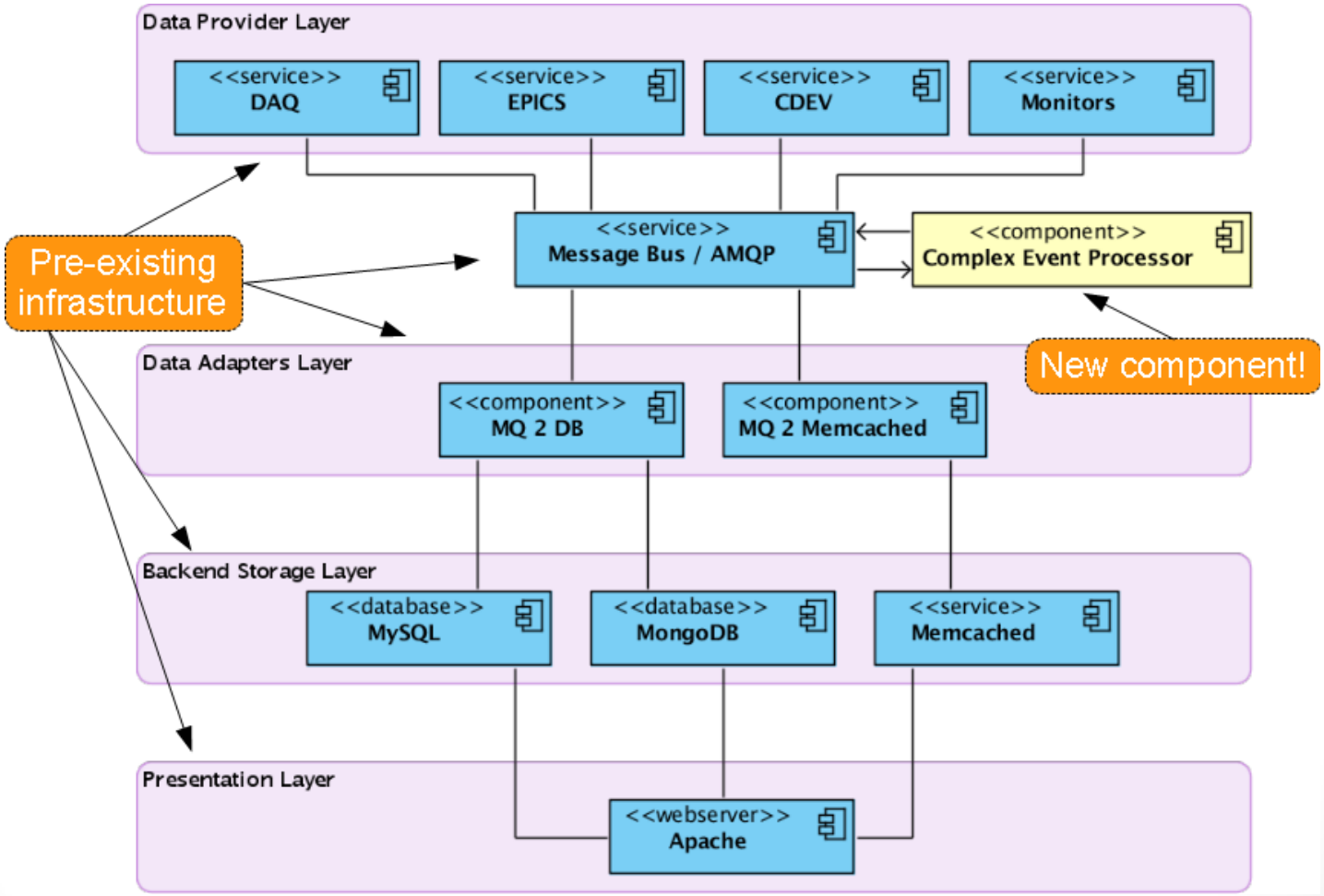
What is Complex Event Processing (CEP) …

- Complex Event Processing (CEP):
  - operates on continuous streams of data coming from many sources
  - understands and manages stream relations (time based, relational syntax, …)
  - can handle high event rate
  - Able detects patterns in data
  - produces output event streams or individual events

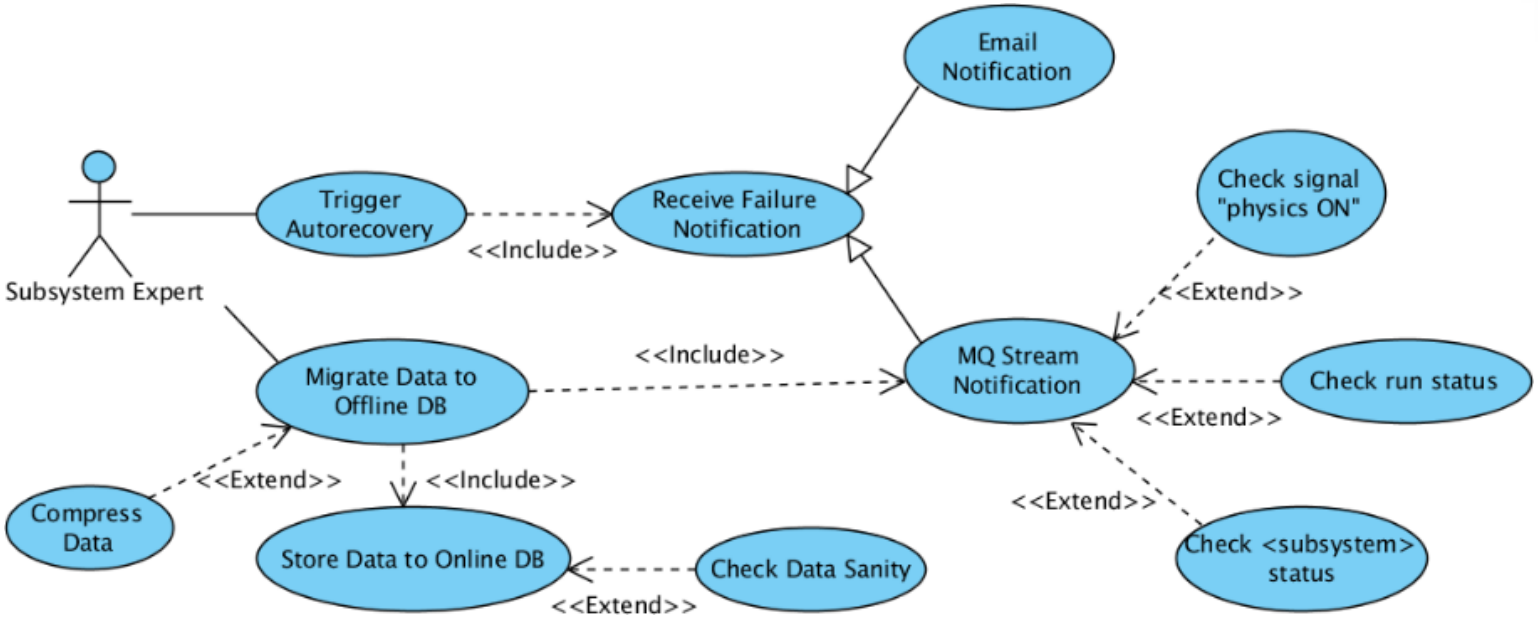
- CEP service includes:
  - input/output data broker (MQ, REST, WS)
  - event processing engine (persistent queries)
  - stream manager (add/remove sources on a fly)
  - query manager (add/remove queries on a fly)

- STAR applications:
  - unify stream data processing, provide single interface for subsystem experts keeping same three-tier schema
  - allow controlled experiment workflow: shift leader hints, experiment run-time tasks orchestration, alarms etc..

Where doe sit fit? Online example



Use-case – online collection of events / meta-data



STAR Subsystem services located in online domain depend on other subsystem states including Trigger, DAQ and the data exported from collider run-time systems..

Middleware

- **WSO2 Complex Event Processor** (CEP) "… is a lightweight and easy-to-use, 100% open source middleware product, available under Apache License v2.0. WSO2 CEP identifies the most meaningful events within the event cloud, analyzes their impacts, and acts on them in real time. It's built to be extremely high performing and massively scalable.."
- The Complex Event Processor consists of the following components: **CEP Core, Broker Core, Broker Manager**.
- CEP Core contains CEP Buckets which are instances of back-end CEP runtime engines (**Esper**, **Fusion**, **Siddhi**) that process events, and Data Converters for converting events from Map, XML, and Tuple types to back end CEP engine's event type. Total processing on received events and triggering of new events happen at the back end CEP runtime engine of each bucket.
- There are four types of brokers which are Local, WS-Event, JMS/JMS-qpid and Agent. These brokers are responsible for receiving and publishing event on **Thrift, SOAP, REST, and JMS** transports.

Summary & Outlook

- **Cloud Logger**
  - Refactored and redesigned UCM toolkit
  - Scalable multi-tier architecture, based on a combination of messaging infrastructure and RESTful service approach – designed to work everywhere from intranet to the large Cloud installations
  - Improved web interface, schema-free database backend

- **Complex Event Processing**
  - CEP capability has been integrated into STAR Online
  - Event Processor is plugged into the existing Message-Queuing bus, no extra hardware required
  - Allows to employ stream processing techniques in online domain, starting with basic error detection and notification features to acquire experience
  - Such processing could also be used to detect issues in user processing tasks (WiP)