MONITORING THE CMS DATA ACQUISITION SYSTEM

G. Bauer [7], B. Beccati [3], U. Behrens [2], K. Biery [6], A. Brett [6], J. Branson [5], E. Cano [3], H. Cheung [6], M. Ciganek [3,] S. Cittolin [3], J. A. Coarasa [3,5], C. Deldicque [3], E. Dusinberre [5], S. Erhan [4], F. F. Rodrigues [1], D. Gigi [3], F. Glege [3], R. Gomez-Reino [3], J. Gutleber [3], D. Hatton [2], J-F. Laurens [3], C. Loizides [7], J. A. Lopez Perez [3,6], F. Meijers [3], E. Meschi [3], A. Meyer [2,3], R. Mommsen [6], R. Moser [3,8], V. O'Dell [6], A. Ohb [3], L. B. Orsini [3], V. Patras [3], C. Paus [7], A. Petrucci [5], M. Pieri [5], A. Racz [3], H. Sakulin [3], M. Sani [5], P. Schieferdeckerc [3], C. Schwick [3], J. F. Margaleff [7], D. Shpakov [6], S. Simon [5], K. Sumorok [7], M. Zanetti [3]

[1] CEFET, Rio de Janeiro , Brazil; [2] DESY, Hamburg, Germany; [3] CERN, Geneva, Switzerland; [4] UCLA, Los Angeles, California, USA [5] UCSD, San Diego, California, USA; [6] FNAL, Chicago, Illinois, USA; [7] MIT, Cambridge, Massachusetts, USA; [8] Technical University of Vienna, Austria

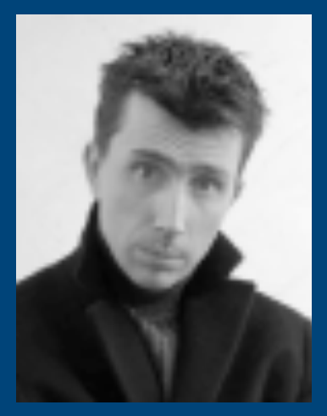
ABSTRACT

The CMS data acquisition system comprises of O(20000) of interdependent services that need to be monitored in near real-time. The ability to monitor a large number of distributed applications accurately and effectively is of paramount importance for operation. Application monitoring entails the collection of a large number of simple and composed values made available by the software components and hardware devices. A key aspect is that detection of deviations from the specified behavior is supported in a timely manner. This is a prerequisite to take corrective actions efficiently. Given the size and time constraints, efficient application monitoring is an interesting research problem. We propose an approach that use the emerging paradigm of Web-service based eventing systems in combination with hierarchical data collection and load-balancing. Scalability and efficiency are achieved by a decentralized architecture, splitting up data collections into regions of collections. An implementation following the presented scheme is deployed as monitoring infrastructure of the CMS experiment at the Large Hadron Collider. All services in this distributed data acquisition system are providing standard web service interfaces via XML, SOAP and HTTP. Continuing on this path we adopted WS-* standards implementing a monitoring system layered on top of the W3C standards stack. We designed a load-balanced publisher/subscriber system with the ability to include high-speed protocols for efficient data transmission and serving data in multiple data formats.

REFERENCES

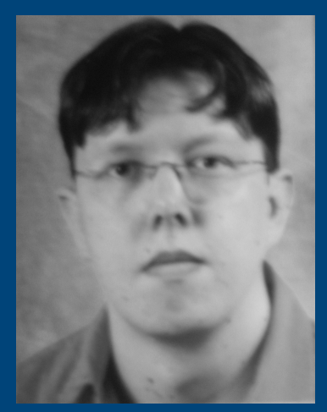
1. CMS, *http://cms.cern.ch, http://cms.web.cern.ch*
2. SOA, *http://soablueprint.com/practitioners_guide*
3. Booth, D. et al. 2004, Web Service Architecture, *http://www.w3.org/TR/ws-arch*
4. Reference Model for Service Oriented Architecture 1.0, OASIS Standard, October 2006, *http://docs.oasis-open.org/soa-m/v1.0/*
5. XDAQ Data acquisition framework, *http://cern.ch/xdaq*
6. Gutleber, J. et al. 2005, Proc. of the 10th Intl. Conf. Accelerator and Large Experimental Phys. Control Sys., Geneva, Switzerland, HyperDAQ Where Data Acquisition Meets the Web.
7. Gutleber, J., Orsini L., 2000, Proc. of the IEEE Intl. Conf. on Cluster Comp., Chemnitz, Germany, IEEE, Architectural software support for processing clusters.
8. Gutleber, J., Murray S., Orsini L., 2003, Elsevier Comp. Phys. Comm. 153(2):155-163, Towards a homogeneous architecture for high-energy physics data acquisition systems.
9. Guttman, E., Perkins, C., Vaizades, J., Day, M. 1999, Sevice Location Protocol, Version 2, *http://www.ietf.org/rfc/rfc2608.txt*.
10. Adobe Flex 3 Rich Internet Applications, *http://www.adobe.com/products/flex/*

CONTACTS



Name: Luciano Orsini
Organization: CERN
Email: Luciano.Orsini@cern.ch
Phone: +41227671615
Web: http://xdaq,web,cern.ch

Name: Johannes Gutleber
Organization: CERN
Email: gutleber@cern.ch
Phone: +41227671536
Web: http://xdaq,web,cern.ch



Name: Roland Moser
Organization: CERN
Email: Roland.Moser@cern.ch
Phone: +41227670808
Web: http://xdaq,web,cern.ch

Computing in High Energy and Nuclear Physics
Prague | Czech Republic | 21 - 27 March 2009

INTRODUCTION

Monitoring the CMS [1] data acquisition system spans all tasks to retrieve, collect and display information used to track the status and operation as well as processing of errors and alarms in a uniform manner. The system is characterized by a large number of hosts and applications. In addition to all traditional requirements that specify the monitoring tasks, scalability requirements are a key concern that pervades all aspects of the system design. Scaling requirements along several dimensions are imposed onto the on-line monitoring infrastructure:

Numerical scalability refers to the ability to seamlessly perform operation with an increased number of users, resources, and services.

Geographical scalability refers to the ability to perform the same identical function regardless of the physical resource location.

Administrative scalability is achieved if the system is managed in the same way even if it encompasses multiple administrative domains. This includes network boundaries, physical computers and mapping of applications to resources.

Functional scalability refers to the ability to accommodate additional functionality.

The proposed infrastructure fits these needs by providing a set of expandable and reusable solutions allowing use of the the monitoring and alarming system for development, test and operation scenarios.

ARCHITECTURE AND DESIGN

The infrastructure is based on **service oriented architecture**[2,3,4], in which a 3-tier structured collection of communicating components cooperate to perform the monitoring task. The universal application connectivity, that makes every monitoring and application services inter-communicating is based on the **XDAQ** [5,6,7,8] middleware. As shown in **Figure 3**, the system builds upon a scalable **publisher-subscriber service** consisting of a pool of **eventing** applications orchestrated by a load balancer called **broker**. The DAQ applications act as data producer through **sensor** services to publish monitoring data. Similarly **sentinel** services are used to report errors and alarms. Other services for processing, storing, filtering and transforming the information express their interest by selectively subscribing to eventing services. Presentation components can either subscribe or directly retrieve monitoring data from the required provider services. All services are re-locatable and run independently of each other without a need for external control. Communication among services is established through a rendez-vous mechanism with the help of **discovery services** facilities [9]. The **heartbeat** service keeps track of all running services and DAQ applications.

DATA COLLECTION

This is the method by which any data tuples defined for the data acquisition system are retrieved from the distributed applications, merged and made available in various standard formats. All metrics are treated using a uniform, table based data format throughout the whole processing chain, see **Figure 2**.
Table definitions enumerating all data items, called **flashlists**, are specified in XML. Flashlist specifications reliably identify the content for merging, tracking and analysis with additional information, including timestamps, version and application identification (URI, URL, UUID, IP and others) fields. These data are inserted by the framework transparently to the application software.
Data collection is initiated in either of two ways at the sources: push from the application or pull according to a configured time period. Merging of distributed tables is performed in one or more steps by a service called **collector**. A load balanced pool of data collectors copes with the data traffic.
The data so collected is served to user interface applications on request in JSON, XML, CSV and SunRPC binary format by the **live access services** over HTTP protocol.

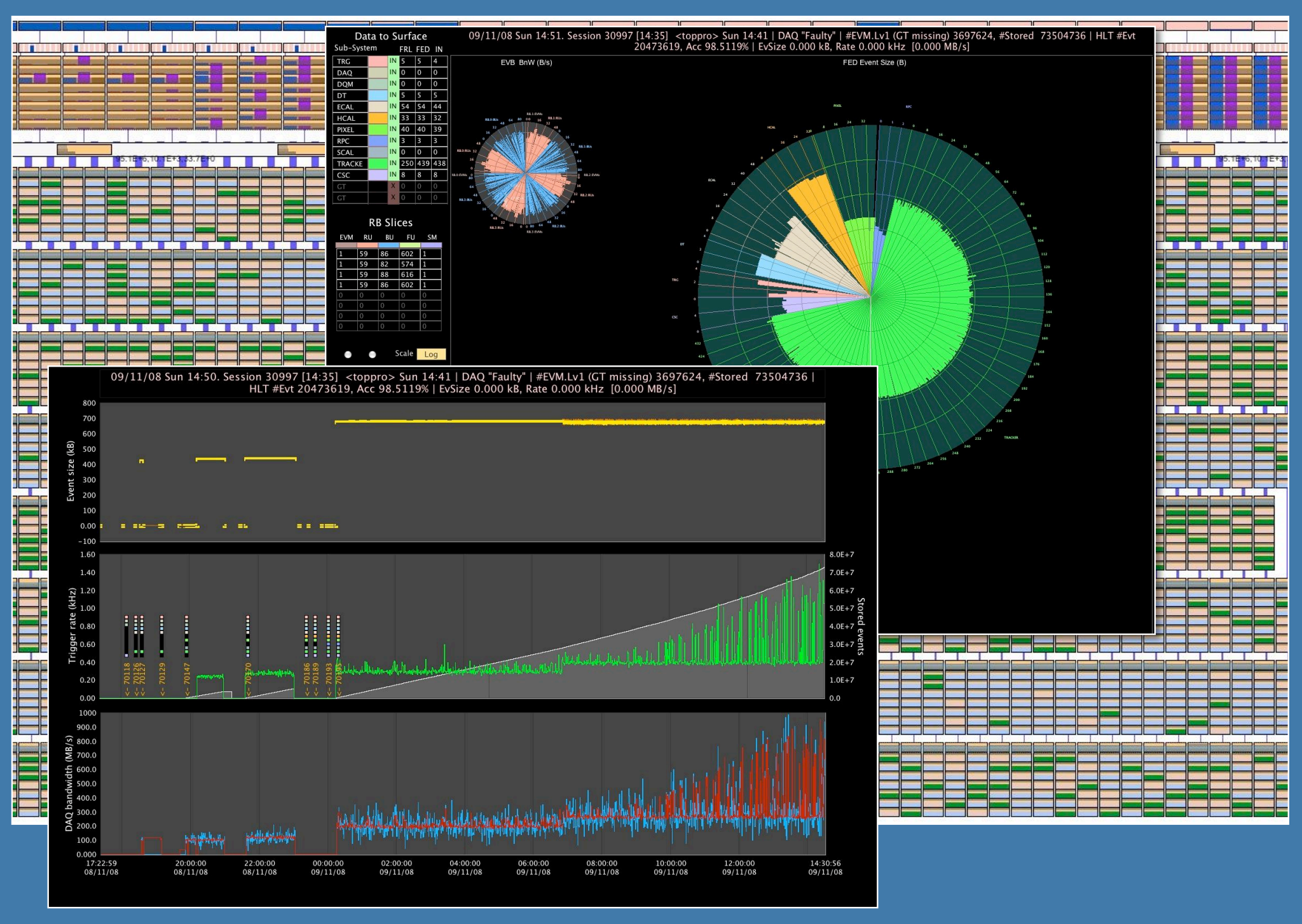


Figure 1. DaqMon (Labview). Layout of the running system with all nodes and their states, history and current status of data flow elements

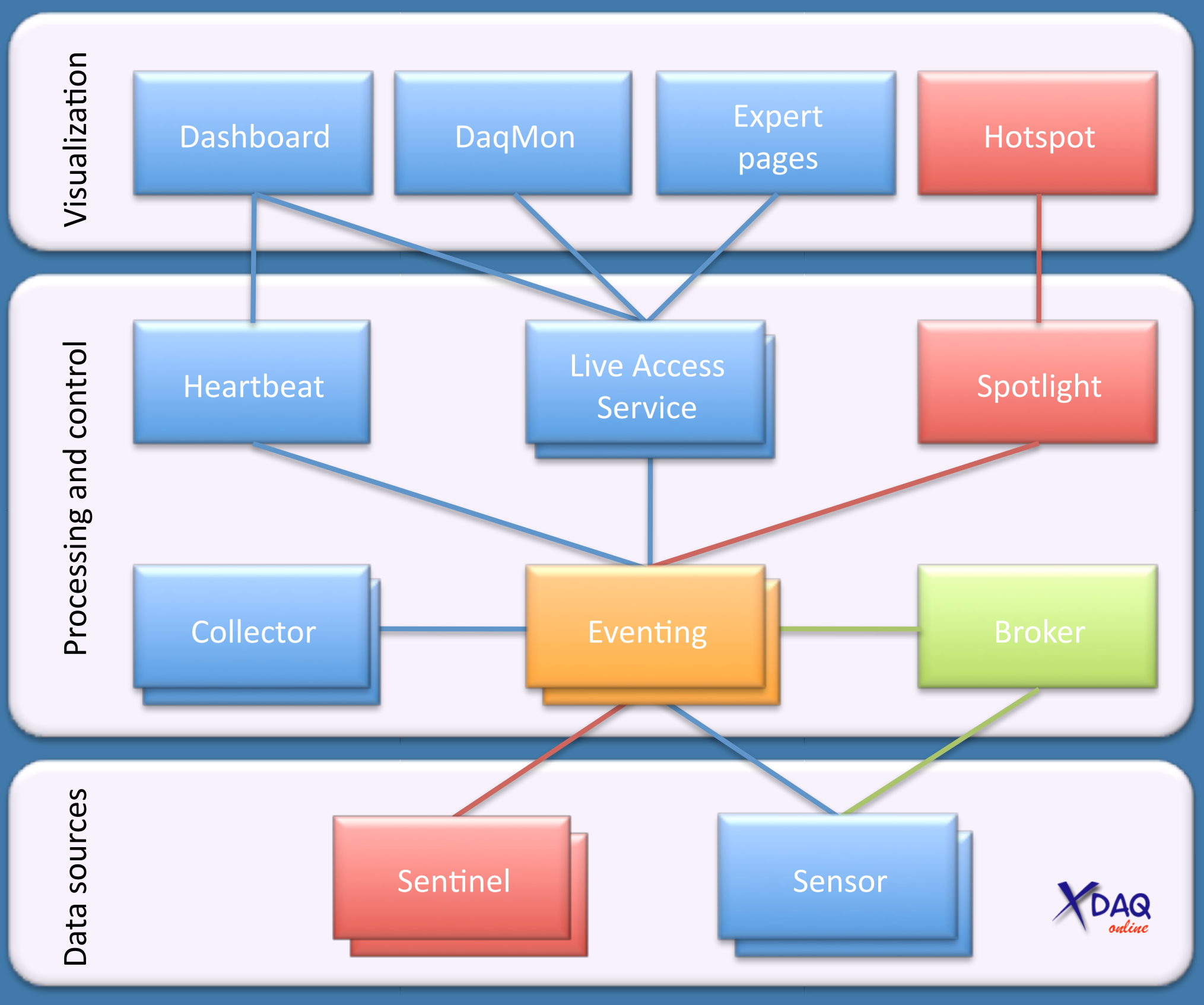
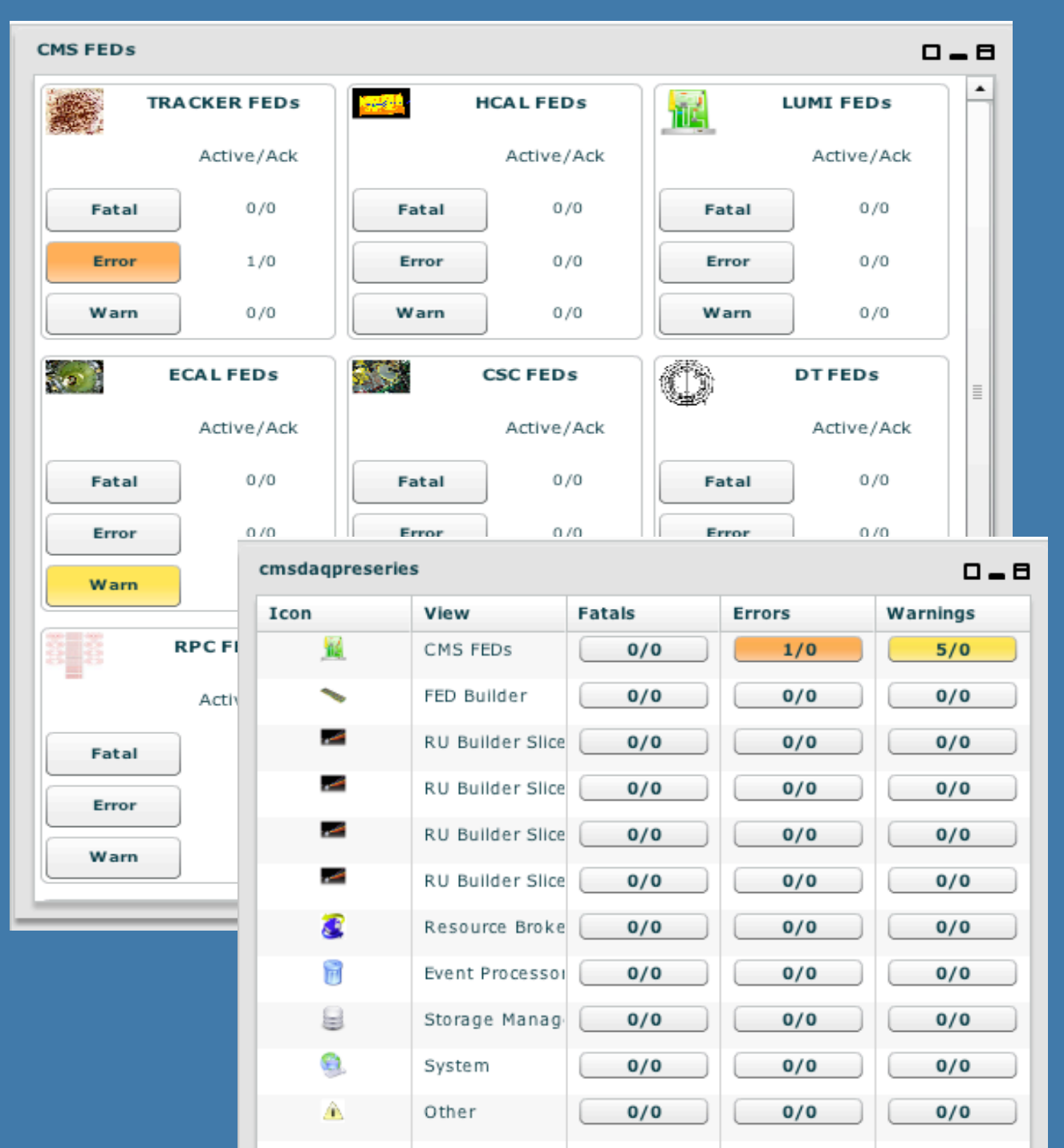


Figure 3. Architecture



CMS FEDs

TRACKER FEDs	Active/Ack	HCAL FEDs	Active/Ack	LUMI FEDs	Active/Ack
Fatal	0/0	Fatal	0/0	Fatal	0/0
Error	1/0	Error	0/0	Error	0/0
Warn	0/0	Warn	0/0	Warn	0/0

ECAL FEDs	Active/Ack	CSC FEDs	Active/Ack	DT FEDs	Active/Ack
Fatal	0/0	Fatal	0/0	Fatal	0/0
Error	0/0	Error	0/0	Error	0/0
Warn					

RPC F	Acti
Fatal	
Error	
Warn	

cmsdaqpreseries

Icon	View	Fatals	Errors	Warnings
	CMS FEDs	0/0	1/0	5/0
	FED Builder	0/0	0/0	0/0
	RU Builder Slice	0/0	0/0	0/0
	RU Builder Slice	0/0	0/0	0/0
	RU Builder Slice	0/0	0/0	0/0
	RU Builder Slice	0/0	0/0	0/0
	Resource Broke	0/0	0/0	0/0
	Event Processo	0/0	0/0	0/0
	Storage Manag	0/0	0/0	0/0
	System	0/0	0/0	0/0
	Other	0/0	0/0	0/0

Figure 5. Hotspot viewer. Example of errors and alarms report according to two different perspectives of the system. Errors and alarms are associated to elements of the system model and displayed according to their severity levels.

ERRORS AND ALARMS

DAQ applications have the capability to asynchronously notify exceptional conditions using a uniform data format to the monitoring infrastructure.
Two different scenarios can be identified. Applications that detect persistent deviations from the normal system behavior can report **errors**. A deviation may also be transient. Therefore an **alarm** is fired and eventually revoked when the asserted condition is resolved.
Reporting errors and alarms is performed through **sentinel** services that take care of routing notifications, guaranteeing delivery and preventing flooding the system. All reports are recorded by a persistency service called **spotlight** that keeps the history of all events. This allows playing back the occurred process. Errors and alarms are visualized by the **hotspot** [10] facility that maps them to the graphics according user defined models of the system.
The tool offers different views on the model such as tree navigation, heat maps, tables and scrolling terminals, see **Figure 5**.

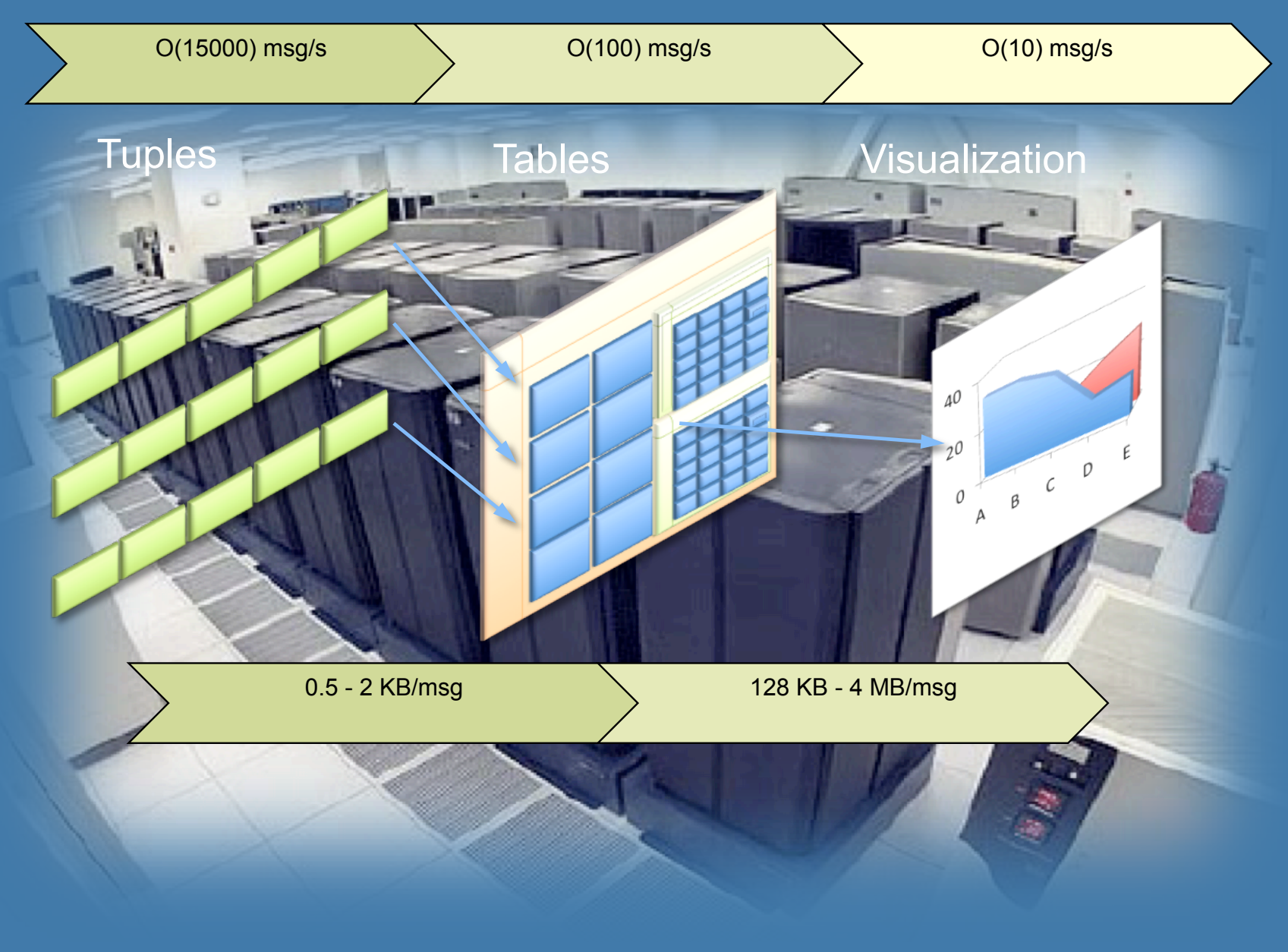
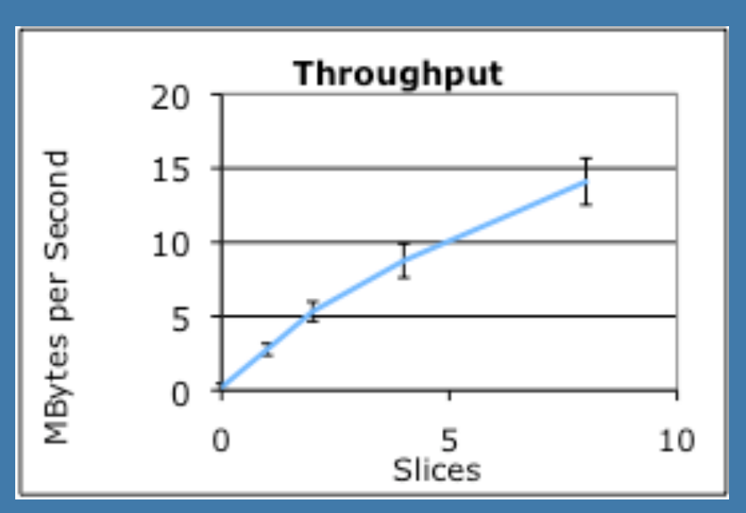
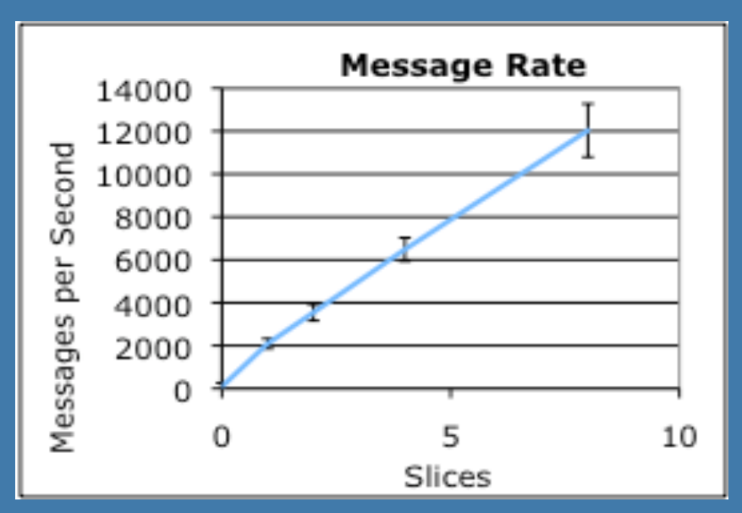


Figure 2. Data life cycle from monitorable sources to user display. Tuples from all applications are merged into hash tables according to configuration.

BENCHMARKS

The two plots below give **scalability** measurements for different system sizes in terms of total message rate and throughput. Increasing the size means adding slices starting from ≈800 applications on ≈150 computers to 5500 applications on 1000 computers. Standard deviation grows with the system size.
The achieved performance allows running the system at the required update rate of **1 Hz for all data sources**.
The current system collects about 20 different flashlists and all updated values can be synchronized within 1 second. The latency for each report depends on the number of collection steps. It has been measured to be within one second.


Throughput
MBytes per Second
Slices


Message Rate
Messages per Second
Slices