Effective runtime monitoring of distributed event-based enterprise systems with ASIA

Sebastian Frischbier*, Erman Turan*, Michael Gesmann†, Alessandro Margara‡, David Eyers§, Patrick Eugster¶, Peter Pietzuch‖, Alejandro Buchmann*
*Technische Universität Darmstadt, *lastname*@dvs.tu-darmstadt.de
†Software AG, Michael.Gesmann@softwareag.com
‡University of Lugano (USI), alessandro.margara@usi.ch
§University of Otago, dme@cs.otago.ac.nz
¶Purdue University, p@cs.purdue.edu
‖Imperial College London, prp@doc.ic.ac.uk

***Abstract*—Cyber Physical Systems (CPS), interconnected smart devices in the Internet of Things (IoT) and other data sources are increasingly bridging the gap between the physical and digital world by providing fine-grained data about real-world events. Enterprise software systems are adopting the paradigm of event-based systems (EBS) to enable them to react to meaningful events in a timely manner. Smart supply chains fusing dynamic sensor data with information provided by backend-systems are one such example of event-based enterprise systems.**

Monitoring their global state in an effective way for runtime governance remains an open research challenge: providing the required type of information while trading off precision for costs. We previously introduced application-specific integrated aggregation (ASIA) as a means for collecting metadata in distributed event-based systems. In this paper, we show how ASIA can support IT Service Management in monitoring and governing decentralized event-based enterprise systems at runtime. We present a dashboard based on industry-strength technology as proof of concept and discuss how to integrate usage statistics provided on-the-fly by ASIA into metrics for runtime governance. We evaluate our monitoring approach in terms of performance, scalability and precision.

I. Motivation

In the Internet of Things (IoT), a multitude of data sources offer continuous information about events taking place in the physical world. Prominent examples are temperature readings about temperature-sensitive freight or geographical coordinates of trucks delivering goods. Enterprise software systems today have to bridge the gap between the physical and the digital world by fusing the information provided by Cyber Physical Systems (CPS) and other data sources with metadata provided by backend systems [3].

Event-based systems (EBS) within an enterprise application landscape complement service-oriented architectures (SOA) to leverage streams of dynamic real-time information and react to meaningful events in a timely manner [2]. For example, smart supply chain management (SCM) systems can automatically redirect delivery routes of cargo containers or trigger processes to replenish goods if they detect delays along the supply chain based on incoming notifications; financial trading applications decide to buy or sell company shares based on real-time news feeds; and data centre management systems scale and reassign resources based on detected usage patterns [17]. In these examples, services in a SOA are invoked by components of an EBS that receive or detect meaningful events. Service invocations, in turn, can result in meaningful changes to a system, triggering components of an EBS to publish events.

EBS are anonymous, information-centric networks. They consist of loosely-coupled software components with different roles that communicate asynchronously using messages: *Publishers* are components that publish notifications if they have detected a specific event taking place. *Subscribers* are components that want to be notified about specific events. Subscribers and publishers are fully decoupled by a *notification middleware* that *pushes* notifications from publishers to subscribers as soon as they are published. Dependencies between components are formed dynamically at runtime, based on the type, quality-related properties or content of notifications [11]. For example, notifications about `temperatureEvent` with content $temperatureCelsius = 30$ and a confidence of 95% are published at a rate of 10 events/second.

As EBS are augmenting core parts of enterprise software systems alongside SOA, runtime governance based on service level agreements (SLAs) has become a key issue for IT Service Management (ITSM) [8]. A key component for any runtime governance activity is effective runtime usage monitoring to identify hotspots or bottlenecks on-the-fly, as opposed to ex-post analysis based on parsed log files [1]. Effective monitoring balances the required data quality of monitoring information with the costs of capturing that information within a given time constraint. Data quality refers to the granularity, precision, consistency or freshness of monitoring updates while costs are measured in terms of performance and traffic overhead [17].

Runtime monitoring for EBS refers to monitoring the population and dynamics of the system: (1) the number of publishers and subscribers that are active for notifications of a certain type, property or content; (2) the rates at which those notifications are supplied and requested. This exceeds common packet-level monitoring of network traffic as the individual information itself has to be observed—for example monitoring the number of publishers providing temperature data about

cargo container #50 with at least 75% confidence.

Providing effective monitoring for decentralized and distributed EBS remains an open research topic due to the inherent anonymity and scalability of EBS. Current approaches have limited effectiveness as they require the deployment of additional monitoring overlays (e.g. SDIMS [26] or Adam2 [23]) that provide a fixed set of available metrics with limited granularity. Traffic overhead and additional effort for operation and maintenance increases monitoring costs [9].

We propose a new approach to effectively monitor large-scale distributed EBS based on the concept of *application-specific integrated aggregation* (ASIA). Our approach provides fine-grained runtime monitoring metrics about the population and dynamics of an EBS without compromising performance and scalability. Users can deploy their own metrics and individually balance the measurement costs with the freshness and precision of the reported monitoring information. Users of our approach do not have to frequently *pull* information about the current state of the system. Instead, they are *informed* proactively only if the state of the system has changed to a degree that they have defined as being significant for them.

Our approach allows users to individually specify the *granularity* of the metrics they want to be updated about and the *precision* of these updates. Granularity refers to what is measured (e.g., the number of subscribers for any `temperatureEvent` versus the number of subscribers only interested in `temperatureEvent`s for cargo container #50 at confidence $\geq$78%). Precision specifies the degree to which the reported system state differs from the true value. For example, a user wants to be notified only if more than ten subscribers for `temperatureEvent` have left or joined the system; smaller fluctuations in membership are not considered to be significant. Our approach exploits such relaxations on information precision in a novel way to limit the propagation of unnecessary updates within the network and to clients.

This paper makes the following contributions:

1) we describe a novel concept for effectively monitoring large-scale distributed EBS at runtime based on the notion of application-specific integrated aggregation (ASIA) that we have introduced in previous work [9];
2) we show how users of our approach can individually balance the precision and freshness of the monitoring information they receive against the costs for providing it by choosing different levels of *imprecision* applied to the monitoring updates reported to them;
3) we evaluate our monitoring approach in terms of performance, scalability and precision using a distributed deployment on 32 physical machines;
4) we present a dashboard based on industry-grade software technology to visualize key metrics about the global state of a distributed EBS on-the-fly.

The remainder of this paper is structured as follows: Section II provides background information on distributed EBS and ASIA. Section III focuses on challenges for runtime governance of event-based enterprise software systems using an SCM example. The architecture and implementation of the ASIA dashboard built with Software AG Mashzone and Java enterprise technology is described in Section IV. We evaluate our monitoring approach in terms of performance, scalability and precision in Section V. We discuss related work in Section VI. We summarize our findings in Section VII and outline steps necessary to apply the FIT-metric [7] to event-based applications as part of ongoing and future work.

II. Background: Distributed EBS and ASIA

A. Event-based systems in a nutshell

Event-based systems (EBS) are reactive systems designed around the concept of *events*. An event is defined as a significant change of state in the physical or digital environment of a system [13]. Publishers report each detection of an event by *publishing* a notification that describes the event; they announce the events that they are going to report on using *advertisements*. Subscribers express their interest by *subscribing* to notifications that match a class of events, a set of properties, or specific content; subsequent lack of interest is expressed by *unsubscribing* from already subscribed events. Subscribers and publishers use an API offered by the notification middleware [21]. The middleware transparently matches subscriptions to advertisements and routes notifications to interested subscribers. Publishers and subscribers are unaware of each other and are anonymous. The middleware can consist of a single, centralized message broker or a distributed and decentralized network of brokers as illustrated in Fig. 1. Each broker requires only local knowledge about its directly connected neighbors (c.f. Fig. 1).

This is beneficial for scalability but complicates the task of maintaining a global view on the state of the system: local state information has to be aggregated and synchronized between brokers. The dynamic nature of EBS means that the aggregated data continuously changes over time, resulting in a large number of synchronization messages. Current solutions for providing aggregate information include the use of centralized messaging systems, group communication systems that provide a form of membership view, or straightforward extraneous direct communication between components. All of these approaches, however, hamper scalability. Distributed aggregation systems (e.g. Astrolabe [25], SDIMS [26] and Adam2 [23]) are scalable but require the deployment of a stand-alone system alongside an EBS. This has multiple drawbacks: it increases the overall system complexity; results in redundant network traffic; and potentially leads to inconsistencies with the state of the monitored EBS.

B. ASIA: aggregate metadata effectively in distributed EBS

We propose a mechanism to effectively monitor the system state of distributed decentralized broker networks using the concept of *application-specific integrated aggregation* (ASIA).

ASIA allows the various components of the distributed system to collect (aggregated) information about each other, without affecting the scalability and performance of the system. Instead of adding a monitoring overlay, ASIA dynamically *integrates* monitoring functionality into the broker network at

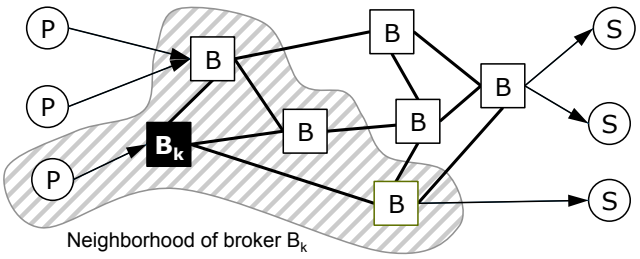

Fig. 1. Distributed decentralized EBS with publishers (P) & subscribers (S). Brokers (B) need only local knowledge about direct neighbors (hatched area).

runtime, using an approach that is inspired by aspect-oriented programming (AOP). Integrating ASIA *into* the broker network provides scalability, performance and precision benefits: (i) it eliminates the cost of creating and maintaining a separate infrastructure; (ii) it enables piggybacking of information on existing messages; (iii) it facilitates use of the local knowledge of brokers, improving the quality of the provided information.

We have successfully implemented and evaluated a prototype within the distributed REDS open-source middleware,[1] focussing on the overhead of dynamically deploying new aggregations. A second implementation using the industry-grade ActiveMQ[2] infrastructure is operational for a single message broker; support for networks of ActiveMQ brokers is part of ongoing work. We refer to [9] for details about the software engineering perspective of ASIA and its evaluation.

In this paper, we focus on how users and the system can benefit from using ASIA for effective runtime monitoring of large-scale distributed EBS.

At the heart of our approach is the notion of *imprecision*: users can individually define what they consider to be *insignificant* changes they do not want to be informed about. For each monitoring metric they are interested in, users can specify an *imprecision* $\hat{v}$ at runtime; it specifies how far the observed system state is allowed to vary from the most recent report of metric values, before an update is triggered regarding those metrics' values. Our novel contribution is to propagate this relaxation throughout the network of an EBS: imprecision is split up between neighboring brokers and applied by every participating broker in the EBS, minimizing the number of update messages necessary to generate and process. For example, each broker can evenly split the imprecision among all its neighbors when propagating a request for monitoring data. We use imprecision not only to mask insignificant changes to the user when displaying data but we already optimize the generation and processing of monitoring updates within the network to save system resources.

Fig. 2 illustrates the resulting trade-off: requests for monitoring data with high precision result in a large number of update messages as even minor changes are reported. In turn, higher imprecision reduces update messages but leaves users with a coarse-grained representation of the system state.

[1]http://zeus.ws.dei.polimi.it/reds/
[2]https://activemq.apache.org/

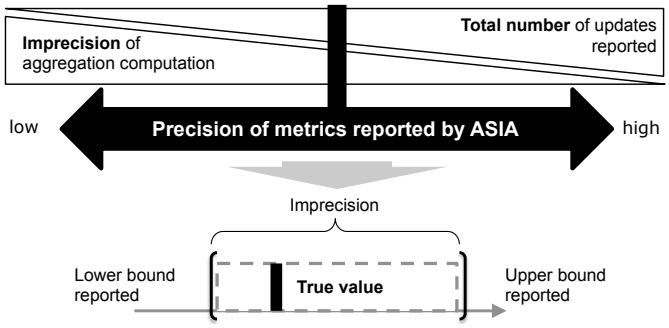

Fig. 2. Imprecision trades data precision with costs for processing updates: upper and lower bounds of monitoring updates enclose the true value.

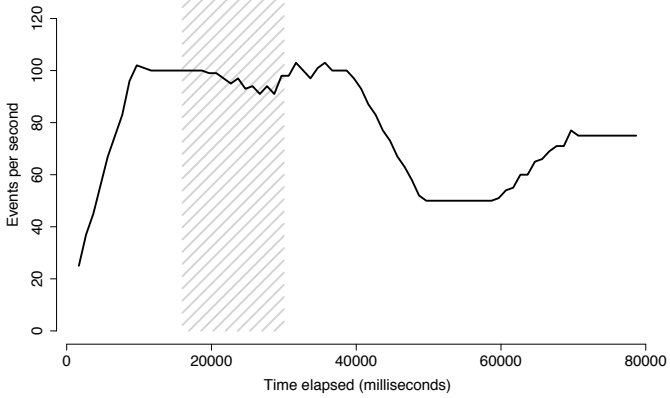

Fig. 3. Example for a changing publication rate: average number of notifications per second published using the workload described in Sect. V.

As a result, updates reported by ASIA to neighbors do not contain a single value for each metric but an *interval* of values (lower bound and upper bound) that encloses the true value for this metric (c.f., Fig. 2, bottom); the size of the interval depends on the chosen imprecision. Updates are generated and sent to neighboring nodes in the network only if the true value *exceeds* the boundaries of the interval that has been reported to that neighbor before. Thus, the system does not have to process unnecessary updates.

For example, the rate at which certain events are published per second ($publicationRate$) changes dynamically over time as shown in Fig. 3. A user might not care about deviations of ± 20 events/second, thus chooses an imprecision of 20. Consequently, the reported updates to the system state reflect the general rate distribution over time but hide insignificant changes. Fig. 4 illustrates this for the workload shown in Fig. 3: the temporarily declining rate (hatched area) is ignored.

III. Runtime governance of EBS with ASIA

We illustrate some challenges for runtime governance of EBS using the following example from the domain of smart supply chain management [14], and indicate how ASIA can help address those challenges.

A. Example: Smart Supply Chain Management

Consider a supply chain with multiple cooperating companies that processes temperature-sensitive chemicals. Com-

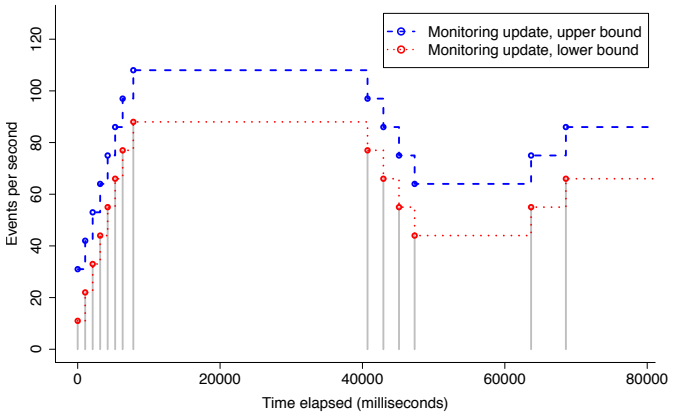

Fig. 4. Higher imprecision hides details: publication rate (c.f., Fig. 3) as perceived by clients using ASIA aggregations with an imprecision of 20.

pany ChemProvider produces the chemicals, companies Carrier1 and Carrier2 ship them to the consuming companies PetroChem and PharmaChem. The chemicals are transported in smart cargo containers provided by ChemProvider. Each cargo container is equipped with a position sensor and a set of wireless temperature sensors to monitor the cargo container they are attached to. Each temperature sensor produces notifications of the type `temperatureEvent` with a property identifying the cargo container (i.e., $ccontainerID = 50$) and a measurement in °C; the position sensor sends notifications of the type `positionEvent` with a property identifying the cargo container and its geographical coordinates. All sensors send their notifications to a message broker located on the cargo-ship or truck they are stored on. Those message brokers are part of the network of either Carrier1 or Carrier2. However, they form a decentralized network of brokers together with brokers controlled by ChemProvider, PetroChem and PharmaChem. At warehouses, checkpoints, ports and highways, passing trucks are automatically registered by traffic monitoring systems, RFID readers or other devices.

All three companies are interested in position updates of their respective shipments to determine the state of supply. Furthermore, they are interested in the containers' current temperatures, measured by at least 3 sensors at a defined minimum rate for quality assurance purposes. This information is required by different business critical enterprise applications with tight SLAs such as systems for enterprise resource planning (ERP) or advanced planning and scheduling (APS).

B. Runtime monitoring metrics

The whole setup can be modeled as a decentralized EBS spanning across multiple companies: sensors attached to the containers advertise and publish updates about position and temperature while the enterprise applications interested in updates subscribe to notifications about containers associated with their shipments. The various devices able to identify trucks are acting as publishers as well. Notifications are forwarded by the distributed broker network across company borders. As part of runtime governance, all parties have to assure that the participating systems are operating efficiently and meet their SLAs.

Focussing on publishers and subscribers in our scenario, IT Service Management at PetroChem and IT Service Management at PharmaChem have to frequently check that their systems that are acting as subscribers are supplied with the information they require at the requested rate by a sufficient number of alternative sources; otherwise an alarm is raised. IT Service Management at ChemProvider remotely switches energy-constrained publishers to a power-saving mode as long as their data is not required (e.g., no active subscriptions or many other active publishers already for the same container).

These tasks require additional runtime monitoring information about the population and dynamics of the system that is usually not available, as every participating party in the scenario controls only a fraction of the decentralized network infrastructure and does not want to share detailed monitoring insights [8]. Aggregations for the whole system, however, can be exchanged across company borders and provided by our approach for the set of event types (E) currently handled:

$$subscriberCount_e \quad \text{\#subscribers active for } e \in E \tag{1}$$

$$publisherCount_e \quad \text{\#publishers active for } e \in E \tag{2}$$

$$publicationRate_e \quad \text{\#events/sec } e \in E \text{ is published} \tag{3}$$

$$subscriptionRate_e \quad \text{\#events/sec } e \in E \text{ is subscribed to} \tag{4}$$

We use these aggregations to define additional key performance indicators (KPIs) for single publishers and subscribers based on the set of event types e they publish ($P_e \subseteq E$) or subscribe to ($S_e \subseteq E$). For example, in our scenario ITSM defines for each publisher j: the total number of subscribers that publisher j is serving across all published types of notifications (5), the relative importance or power of supply (PoS) of publisher j by providing e (6), and the power of demand (PoD) for e compared to the overall demand (7):

$$servedSubscribers_j = \sum_{e}^{P_e} subscriberCount_e \tag{5}$$

$$rPoS_{e,j} = \frac{1}{publisherCount_e} \tag{6}$$

$$PoD_e = \frac{subscriberCount_e}{\sum_e^E subscriberCount_e} \tag{7}$$

IV. A DASHBOARD FOR RUNTIME MONITORING OF EBS

Our approach provides runtime monitoring information about the population and dynamics of an EBS as described in Sect. III-B. We have implemented the prototype[3] of a dashboard to provide a visual summary of the system state by fusing runtime monitoring updates with historic data.

A. Dashboard design: data- and client-centric views

Our dashboard provides a *data-centric* view as well as *client-centric* views for individual publishers and subscribers.

The data-centric EventDataTab summarizes supply and demand for all active events and provides a view of the global

[3]https://www.dvs.tu-darmstadt.de/research/events/asia/.

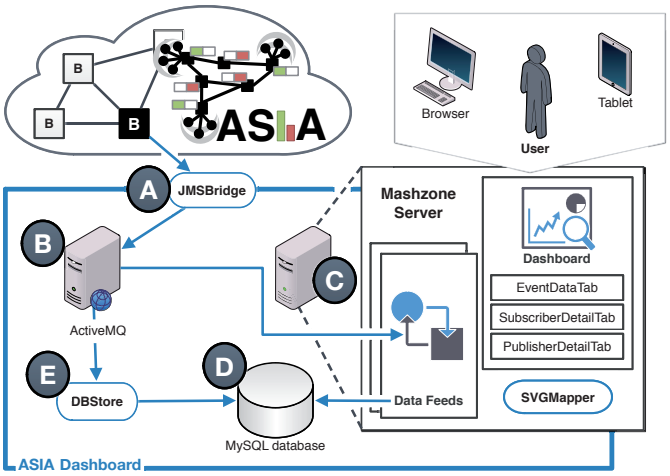

Fig. 5. Architecture of the dashboard prototype using ASIA, Software AG Mashzone and Java enterprise technology.

state of the system. The client-centric PublisherDetailTabs and SubscriberDetailTabs[4] in turn provide drill-down views for event types handled by a specific client.

In addition, an interactive graph representation of the known network topology is provided that facilitates easy navigation to specific publishers (diamond shape) or subscribers (oval shape). Referring to our SCM example, this view might be restricted to applications and message brokers operated by a single company participating in large-scale distribution networks.

An *EventDataTab* view shows monitoring updates delivered on-the-fly about metrics (1)–(4) for each active event type $e \in E$. For each metric, additional *historic data* traces stored in the back-end database can be visualized at the bottom of the tab. Furthermore, we display system-wide metadata notifications (pinpoints) with their timestamp as a news feed and as an overlay over the historic data. This allows users to visually check if specific events (e.g., outages, planned maintenance) have impacted a given metric.

A *PublisherDetailTab* is available for each publisher that is part of the network controlled by the operator of this dashboard. It displays the additional KPIs (5)–(7) for the types handled by this publisher instance, effectively providing a drill-down view on the data available in EventDataTab. Historic data traces can be visualized as well.

The upper and lower bounds reported by ASIA for each metric can be used to implement custom and application-specific tools for process control (e.g., using control or EWMA charts) that trace the history of a metric and use thresholds to trigger actions based on deviations from defined target values.

B. Architecture and implementation using Mashzone

The general architecture of our prototype consists of five components, as illustrated in Fig. 5: updates reported by ASIA about the state of the EBS are forwarded to the dashboard by JMSBridge (A) using ActiveMQ as Java Message Service (JMS) server (B). The Mashzone server (C) provides the views described in Sect. IV-A on a web-based dashboard, and uses JMS to receive runtime updates for metrics. The same updates are stored in a separate MySQL database (D) by DBStore (E) as traces for detailed ex-post analysis. Information about the known system topology can be provided beforehand or on-the-fly. We assume that topology information is available beforehand as part of the asset metadata stored in governance platforms. The interactive topology map is automatically generated from a graph representation provided in the open XML format, GraphML, by SVGMapper.

JMSBridge is connected to ActiveMQ as a publisher and to a message broker of the EBS as a listener, registering for runtime updates about the global state of the system.

Note that we do not require our listener to connect to a specific broker in the EBS—any broker of the system implementing ASIA is able to deliver the required metrics.

The set of monitoring metrics to request together with the required imprecision can be configured automatically or manually. JMSBridge should register for updates about the $publisherCount$ of $\texttt{temperatureEvent}(containerID = 50)$, while allowing for an imprecision of 10, for example.

JMSBridge *is notified* by ASIA whenever the value of the requested monitoring metrics change to a degree that is of interest to the listener (c.f., Sect. II). In Mashzone, the Dashboard Layer used for visualization retrieves runtime updates and historic data in a uniform way using Data Feeds. The dashboard can be accessed by using web browsers in a platform-independent way or by using platform-specific apps.

V. Evaluation

We have evaluated our monitoring approach using the prototype implemented within the distributed REDS open-source middleware. As a workload-generator we have extended the open-source SPEC Research benchmarking tool Fincos[5] to request and receive ASIA aggregation updates.

The experimental evaluation focusses on quality of information (QoI) and quality of service (QoS) aspects of our monitoring approach. We want to:

(A) assess how *precisely* updates on monitored metrics provided by ASIA reflect the current system state—given different imprecisions, number of brokers involved, and workloads that change over time to different degrees;

(B) gauge how well an EBS that applies ASIA adapts to large numbers of clients and high-volume traffic in terms of throughput and delay.

This complements the evaluation related to software engineering presented in [9].

A. Precision of monitoring information (QoI)

We have defined a workload pattern for the rate at which events are published per second ($publicationRate$). It is based on the default workload provided by Fincos, focussing on a rate changing over time. We chose the rate of publication for

[4]For the remainder of this section, we omit a detailed description of SubscriberDetailTab as it corresponds to PublisherDetailTab.

[5]http://research.spec.org/tools/overview/fincos.html

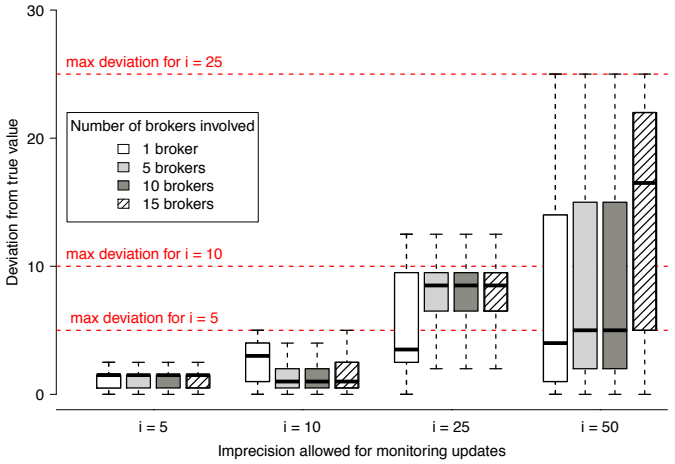

Fig. 6. Precision of monitoring information stays within imprecision boundaries: monitoring updates reported for $publicationRate$ never differ more than the defined imprecision (dashed red) from the true value; random workloads, ten repetitions. Whisker end marks are the max/min value.

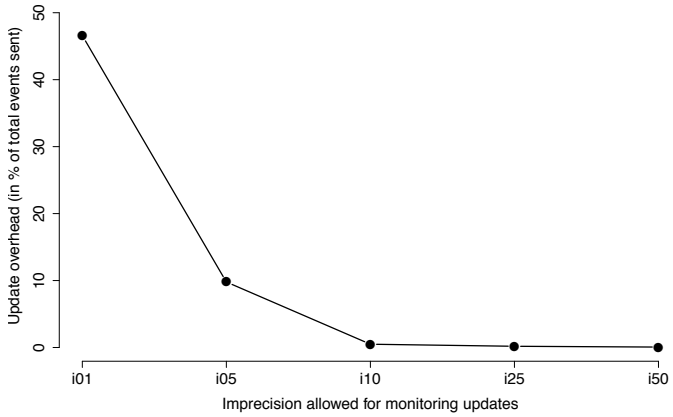

Fig. 7. Imprecision reduces monitoring overhead: reported monitoring updates for the same workload but different levels of imprecision; plotted in proportion to the total number of events processed.

this part of the evaluation as it is a highly dynamic metric. Fig. 3 shows the workload.

We use the same workload with increasing levels of imprecision for *publicationRate* and vary the number of brokers involved from one to 15 brokers. We measure (1) the degree to which ASIA updates vary from the real system state; (2) the number of monitoring updates that have to be processed by the receiving client for each level of imprecision. Tests are repeated ten times for each level of imprecision.

1) Deviation from the true value: Fig. 6 shows that monitoring updates reported by our approach do not differ from the true value more than the imprecision set by the user beforehand, independent of the number of brokers involved. The box-plots show the deviation of the upper and lower bounds reported for randomly generated variations of our workload and changing levels of imprecision. Updates to monitored metrics reported as lower and upper bounds always enclose the true value as shown in Fig. 8 for two samples of the same workload but with different imprecisions.

2) Reduction of updates: With increasing imprecision, fewer updates are necessary to represent the system state as ASIA does not report every change anymore. Fig. 7 shows the percentage of events with piggybacked monitoring updates that have to be processed by a client in proportion to the total number of events processed for the same workload. With increasing imprecision, system and client have to process significantly fewer updates to monitoring metrics the client is interested in, allowing it to free up local resources. A low imprecision of 1 results in 46% of the total number of processed events to be updates for monitoring metrics; an imprecision of 5 already reduces this to 9% while the number of necessary updates drops to less than 1% for an imprecision of 10 (0.49%), 25 (0.18%) or 50 (0.06%).

B. Performance and scalability (QoS)

We compare the impact on throughput and delay caused by our monitoring approach against the performance of an EBS without support for aggregation, also implemented in REDS.

To measure maximum throughput, we increase the publication rate while observing the rate and delay of messages delivered to subscribers. Each client registers for monitoring information about *subscriberCount*, *publisherCount*, *subscriptionRate* and *publicationRate* for randomly selected events.

The delay for delivering notifications to subscribers is a significant performance indicator in EBS. To measure the delays for notifications and update propagation we use two different aggregation functions: $publisherCount$ and $publicationRate$. These functions are the most challenging for ASIA as update information is piggybacked on top of messages, which are most frequent, and thus have a higher likelihood of causing queueing at intermediate brokers. In addition, we set the imprecision to zero—all updates are immediately delivered to all interested clients.

We have deployed our approach on a testbed for distributed systems consisting of 32 physical machines simulating a distributed EBS with up to 1600 clients connected to a total of 16 brokers. Each broker is deployed on a separate machine, while all clients connected to a given broker are hosted on the same machine. The average network latency is 0.2ms.

Fig. 9 shows our results. For both the bare EBS and for ASIA, the throughput initially grows with the publication rate (c.f., Fig. 9 (left)). We observe that the throughput saturates at the same publication rate (i.e., 100 messages per second per agent) and that the maximum throughput value is almost identical for both systems (i.e., 160,000 messages per second in total). Regarding delay, the measured values for an EBS applying ASIA are comparable to those for a bare EBS, both in terms of average and the 95th percentile of the delay. This indicates that, even under an extreme workload and without any imprecision, our approach does not introduce a noticeable overhead to throughput and delay.

Our evaluation shows that we can provide precise runtime monitoring information about the population and dynamics of a decentralized EBS without compromising the performance of the monitored system. Furthermore, our approach allows users to significantly influence the number of monitoring updates

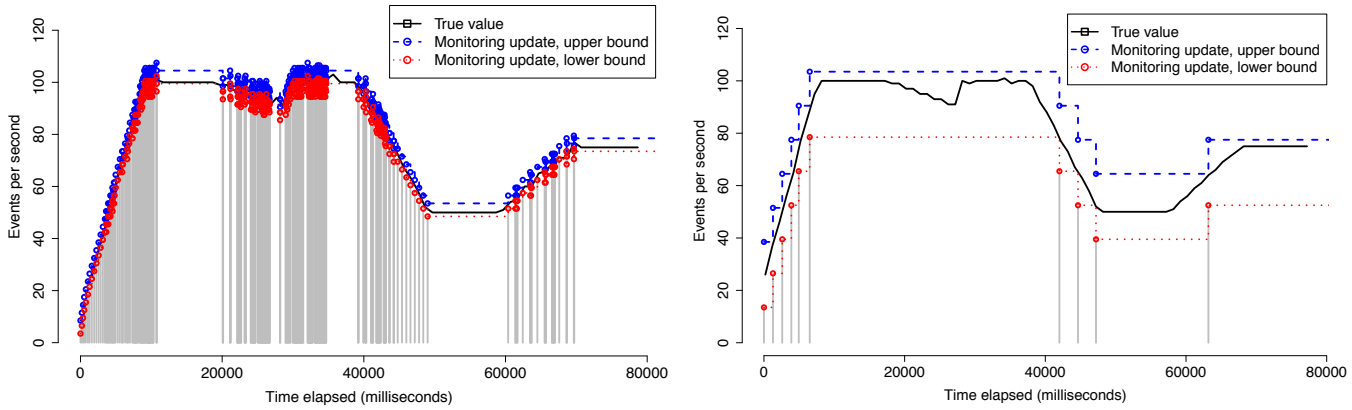

Fig. 8. Monitoring information about $publicationRate$ reported to the user as upper/lower bounds always describes the system state correctly. Increasing imprecision results in fewer updates (gray lines) and a coarse representation; imprecision 5 (left) and 25 (right) are shown for the same workload (c.f., Fig.3).

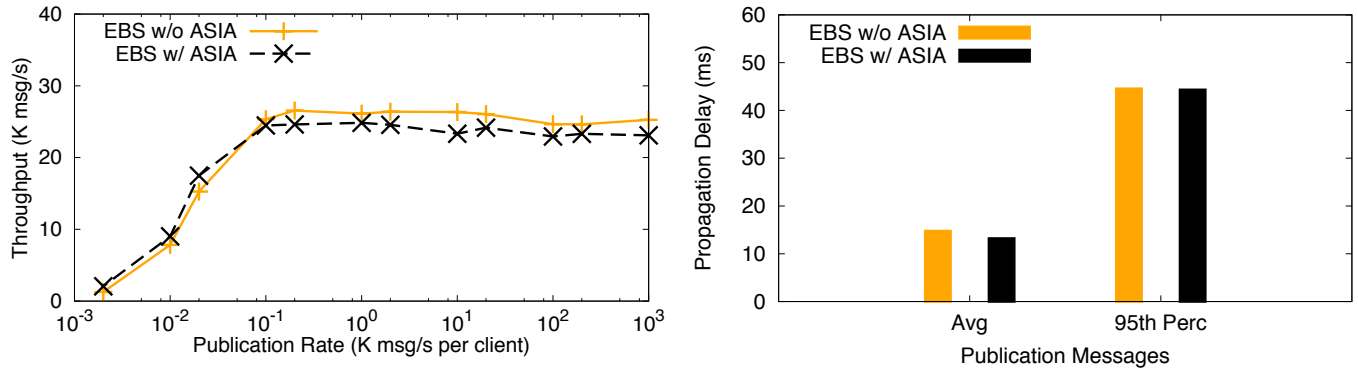

Fig. 9. Maximum throughput for different publication rates (left) and the propagation delay for messages and aggregation updates (right).

they have to process by choosing higher levels of imprecision.

VI. Related work

Several solutions for monitoring large-scale SOA deployments rely on event-based approaches [17]. For example, Michlmayr et al. use EBS inside VRESCo [20]; Smit et al. use STORM[6] to monitor heterogeneous cloud settings [24]; Guinea et al. [12] use SIENA[7] for collecting runtime information about services; Agarwala et al. [1] use EBS to process monitoring updates with QoS constraints. Monitoring approaches for event-based systems, however, are rare. ActiveMQ provides advisory messages to monitor the state of a single queue or topic on a single broker. However, it does not support key metrics such as $publicationRate$ or $publisherCount$. Lee et al. propose ViVa [18] to record all exchanged messages between components of an EBS for debugging and impact analysis. Differently from our approach, which targets runtime governance based on real-time information, ViVa relies on ex-post analysis of log files. Several approaches provide information about the runtime state [25], [19], [4] or stability [15], [5], [16], [23] of an EBS by relying on a separate aggregation system. Astrolabe [25] provides summarization based on user-defined aggregation functions, implemented via a single logical aggregation tree on top of an unstructured peer-to-peer gossip protocol. The authors exemplify the use of Astrolabe in a topic-based EBS, but do not discuss how aggregation is affected by changes in the topology. SDIMS [26] extends distributed hash tables (DHTs) to perform hierarchical aggregation based on attribute types and names. STAR [15] adaptively sets the precision constraints for processing aggregate queries. It is used by Jain et al. [16] to provide network imprecision (NI), a consistency metric for large-scale distributed systems that quantifies a system's stability in terms of currently reachable nodes and number of updates that might have been repeatedly processed due to network failures. In contrast to ASIA, such generic aggregation systems are unable to leverage specific properties of distributed EBS, such as overlay topologies or exchanged messages. Aggregation trees may not match routing trees, resulting in inefficiency and delayed adaptation. Finally, energy efficient in-network aggregation is studied in wireless sensor networks (WSNs) [6]. This is complementary to our work as we do not compute aggregations within the energy-constrained WSN but within the infrastructure of the enterprise software system where energy efficiency is an issue on a different scale. None of these systems support generic, application-specific aggregation or imprecision within the broker network. Neither are they able to piggyback information.

[6]https://storm.incubator.apache.org/
[7]http://www.inf.usi.ch/carzaniga/siena/

VII. Conclusion, ongoing and future work

Event-based systems (EBS) complement service-oriented architectures (SOA) and enable enterprise software systems to react to real-time information. Runtime governance of EBS has become a key issue, stressing the need for efficient and effective runtime monitoring. In this paper, we described a new approach to providing effective runtime monitoring for decentralized EBS using the concept of application-specific integrated aggregation (ASIA). Our approach monitors the global state of the system and allows users to individually balance precision and granularity of monitoring information against the costs of providing it using *imprecision*. We presented a real-time dashboard visualizing the system state and key performance indicators (KPI) for different participants. In our evaluation we have shown that our approach provides precise and fine-grained monitoring information without affecting the monitored system in terms of scalability and performance.

Ongoing and future work focusses on utilizing the monitoring information provided by our approach. We investigate two complementary angles here: the integration into runtime governance processes for EBS and its use as part of a self-adapting, cost-aware transport service for distributed applications in multi-cloud deployments.

Runtime governance for EBS: Together with our industry partners, we are working on integrating monitoring information about EBS into runtime governance metrics by fusing them with backend information that is stored in governance platforms. The FIT metric [7], for example, identifies and ranks *hotspots* in a SOA-based enterprise system landscape by quantifying the criticality of relationships between services. It deduces how *important* any given participating system is for the whole application landscape based on its *Function* (business criticality), *Integration* (impact on other participating systems) and *Traffic* (actual usage). ASIA runtime monitoring information can be used to model the components *Integration* and *Traffic* specifically for EBS components when augmented with metadata stored in governance platforms. For example, the *Integration* of publisher j could be modeled using $servedSubscribers_j$ and $rPoS_{e,j}$ weighted by the importance of each event type e as defined by ITSM; *Traffic* could weight the publication rate of j for e with the global rate that e is produced and consumed by others. Extending the FIT metric to EBS in general is part of future work, addressing the design and calibration of the data model, weights and thresholds.

Self-adaptive, cost-aware transport: Having information about the global state of a distributed system available is a crucial prerequisite for building self-adaptive systems [22]. We use the real-time monitoring information provided by our approach as part of a self-adaptive and cost-aware middleware that optimizes the operation of distributed applications deployed in multi-cloud environments [10].

Security and privacy aspects are important but orthogonal to our approach and currently out of scope.

VIII. Acknowledgements

Funding by German Federal Ministry of Education and Research (BMBF) under research grants 01 S12054 and 01 C12S01V.

References

[1] S. Agarwala, Y. Chen, D. Milojicic, and K. Schwan. QMON: QoS-and utility-aware monitoring in enterprise systems. In *ICAC*, 2006.

[2] S. Appel, S. Frischbier, T. Freudenreich, and A. Buchmann. Eventlets: Components for the integration of event streams with SOA. In *SOCA*, 2012.

[3] A. Buchmann, H.-C. Pfohl, S. Appel, T. Freudenreich, S. Frischbier, I. Petrov, and C. Zuber. Event-Driven services: Integrating production, logistics and transportation. In *SOC-LOG*, 2010.

[4] A. K. Y. Cheung and H. A. Jacobsen. Publisher placement algorithms in content-based publish/subscribe. In *ICDCS*, 2010.

[5] G. Cugola, M. Migliavacca, and A. Monguzzi. On adding replies to publish-subscribe. In *DEBS*, 2007.

[6] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi. In-network aggregation techniques for wireless sensor networks: A survey. *IEEE Wireless Commun.*, 14(2):70–87, 2007.

[7] S. Frischbier, A. Buchmann, and D. Pütz. FIT for SOA? Introducing the F.I.T. – metric to optimize the availability of service oriented architectures. In *CSDM*, 2011.

[8] S. Frischbier, M. Gesmann, D. Mayer, A. Roth, and C. Webel. Emergence as competitive advantage – engineering tomorrow's enterprise software systems. In *ICEIS*, 2012.

[9] S. Frischbier, A. Margara, T. Freudenreich, P. Eugster, D. Eyers, and P. Pietzuch. Aggregation for implicit invocations. In *AOSD*, 2013.

[10] S. Frischbier, A. Margara, T. Freundenreich, P. Eugster, D. Eyers, and P. Pietzuch. McCAT: Multi-cloud Cost-aware Transport. In *EuroSys Poster Track*, 2014.

[11] S. Frischbier, P. Pietzuch, and A. Buchmann. Managing expectations: Runtime negotiation of information quality requirements in event-based systems. In *ICSOC*, 2014.

[12] S. Guinea, G. Kecskemeti, A. Marconi, and B. Wetzstein. Multi-layered monitoring and adaptation. In *ICSOC*, 2011.

[13] A. Hinze, K. Sachs, and A. Buchmann. Event-based applications and enabling technologies. In *DEBS*, 2009.

[14] IBM. The smarter supply chain of the future: Insights from the global chief supply chain office study. http://public.dhe.ibm.com/common/ssi/ecm/en/gbe03163usen/GBE03163USEN.PDF, 2010.

[15] N. Jain, D. Kit, P. Mahajan, P. Yalagandula, M. Dahlin, and Y. Zhang. STAR: Self-tuning aggregation for scalable monitoring. In *VLDB*, 2007.

[16] N. Jain, P. Mahajan, D. Kit, P. Yalagandula, M. Dahlin, and Y. Zhang. Network imprecision: a new consistency metric for scalable monitoring. In *OSDI*, 2008.

[17] K. Keeton, P. Mehra, and J. Wilkes. Do you know your IQ? a research agenda for information quality in systems. *ACM SIGMETRICS Performance Evaluation Review*, 37(3):26–31, 2010.

[18] Y. Lee, J. Bang, J. Garcia, and N. Medvidovic. ViVA: A visualization and analysis tool for distributed event-based systems. In *ICSE*, 2014.

[19] S. Meng, S.R. Kashyap, C. Venkatramani, and L. Liu. Remo: Resource-aware application state monitoring for large-scale distributed systems. In *ICDCS*, 2009.

[20] A. Michlmayr, P. Leitner, F. Rosenberg, and S. Dustdar. Publish/subscribe in the VRESCo SOA runtime. In *DEBS*, 2008.

[21] P. Pietzuch, D. Eyers, S. Kounev, and B. Shand. Towards a common API for publish/subscribe. In *DEBS*, 2007.

[22] H. Psaier and S. Dustdar. A survey on self-healing systems: approaches and systems. *Computing*, 91(1):43–73, August 2010.

[23] J. Sacha, J. Napper, C. Stratan, and G. Pierre. Adam2: Reliable distribution estimation in decentralised environments. In *ICDCS*, 2010.

[24] M. Smit, B. Simmons, and M. Litoiu. Distributed, application-level monitoring for heterogeneous clouds using stream processing. In *Future Generation Computer Systems*, 2013.

[25] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21:164–206, May 2003.

[26] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *SIGCOMM*, 2004.