SAMURAI: A batch and streaming context architecture for large-scale intelligent applications and environments

Davy Preuveneers *, Yolande Berbers and Wouter Joosen
iMinds-DistriNet-KU Leuven, Department of Computer Science, Celestijnenlaan 200A, B-3001 Heverlee, Belgium
E-mail: {firstname.lastname}@cs.kuleuven.be

Abstract. Over the past decade intelligent environments have grown in sophistication. Many recent paradigm shifts − such as the Internet of Things (IoT), Ambient Assisted Living (AAL), e-health and telemedicine − envision large distributed networks of intelligent devices, applications and services that are sensitive to the presence of people and responsive to their needs. Cutting edge technologies will autonomously and collectively operate on a growing volume of information arriving at ever increasing velocities to transparently and non-intrusively support users during their activities. Especially the escalating variety of information that applications have to deal with is a non-trivial concern. Making sense out of heterogeneous and pervasive streams of sensor events to anticipate and address the needs of users is a ubiquitous challenge that many interactive context-aware applications in intelligent environments frequently face. Furthermore, software solutions that continuously interpret the tasks and contexts of a variety of individuals with different needs are often faced with scalability concerns.

We present SAMURAI, a batch and streaming context architecture that integrates and exposes well-known components for complex event processing, machine learning, and knowledge representation. SAMURAI builds upon key concepts of the Lambda architecture and big data enabling technologies to achieve horizontal scalability and responsive interaction with its users. Two application cases validate the feasibility and performance of our context architecture, demonstrating near-linear scalability, flexible elasticity and smooth interaction capabilities.

Keywords: context, batch and stream processing, scalability, intelligent applications

1. Introduction

The exponential data growth is an opportunity to build sophisticated intelligent environments. With the advent of trends like big data, smart applications increasingly obtain more useful information about their users and their preferences. With data being volunteered at an unprecedented scale, context-aware computing is becoming a game changer as it allows service providers of a variety of intelligent environment applications to customize their solutions to their users and this with decisions no longer based on speculative presumptions or manually crafted rules and adaptation logic, but rather on data-driven models. Indeed, mobile and wearable computing platforms, like smartphones and smartwatches [38], embed sensor technology to observe acceleration, location, orientation, ambient lighting, sound, imagery [24]. Especially, in the field of m-health and e-health, the use of such wearable devices is becoming more prevalent [31,39] with mobile health applications to monitor a variety of health parameters including posture [25], heart conditions [45], diabetes [3,34], and physical activity [11]. Furthermore, emerging computing paradigms like the Internet of Things (IoT) [4] will further spark sensor technology to become omnipresent in our surroundings, and will promise a continuous data growth.

Indeed, intelligent environments are evolving to open ended large scale and dynamic network infrastructures fueled by low cost, connected and wirelessly

*Corresponding author. E-mail:davy.preuveneers@cs.kuleuven.be

communicating devices that collect data, relay information to one another, process the information collaboratively, and take actions in an autonomic way. However, relevant information about the user is also becoming more complex, heterogeneous, and scattered. With the increased prevalence of mobile applications, the expectations of a frictionless customer experience, and the diversity in a user's computing environments, tapping into this exponential data growth with conventional methods has become an arduous undertaking. Given the unpredictable peaks of high computational cost to collect and process data, being able to make sense of *large* volumes of data with uncertain *veracity* and *value* from a *variety* of context sources in near real-time will become a key differentiator [19] for future intelligent environment solutions.

Sophisticated intelligent applications services should factor in all relevant context information about the user and his situation, but are often faced with the following non-trivial questions on how to effectively unlock the large yet untapped sources of context information:

- Which information will influence application-level decisions for user adaptive behavior?
- Is this information readily available or does it require additional processing before it is useful?
- Can we process this information offline in batch mode or online in a streaming fashion?
- What are acceptable processing latencies to guarantee a smooth interaction and user experience?
- Can we easily reuse the context processing components for other applications?
- Can our context system be distributed and scale out on demand when the workload grows?

The above questions occur frequently across context-aware adaptive applications, and have amplified the need for context-aware computing solutions offered as services that can deal with:

1. Individuals with different needs sharing the same applications, services and infrastructure
2. Heterogeneous context data sources and event types with varying degrees of veracity
3. Loosely structured and distributed event streams collectively adding value to the application

Enabling technologies that provide a *reusable*, *scalable* and *reliable* software solution for the above concerns have all what it takes to become an indispensable foundation for a wide variety of intelligent applications and environments. We are tackling these challenges by extending and enhancing SAMURAI [35], our previous award-winning research on streaming multi-tenant context-management architecture for intelligent and scalable Internet of Things applications. The first version of this system was developed within the frame of the FP7 BUTLER project[1] whose objective was the creation of a horizontal IoT platform supporting several domains of our daily lives – including home, health, smart cities, energy, transport, shopping, etc. – all at once. In this work, we discuss how we redesigned SAMURAI around the basic principles of the *Lambda Architecture* [27], and how we further enhanced and extended our framework with big data technologies to further scale out to growing amounts of context.

After reviewing related work in section 2, we present two use cases in section 3 as motivating examples. Section 4 discusses the design of our batch and streaming context architecture. In section 5 we evaluate the feasibility and effectiveness of the approach. We conclude in section 6 summarizing the main insights and identifying possible topics for future work.

2. Related work

Before we dive into the contributions of our work, we briefly discuss relevant state-of-the-art on activity recognition, big data enabling technologies and scalable learning and prediction algorithms.

2.1. Behavior-awareness and activity recognition

Especially in the area of home-care applications and Ambient Assisted Living [12] for the elderly, automatic discovery and classification of daily activities plays a key role [51,40] to anticipate the kind of assistance they need or to detect the occurrence of abnormal events (such as a fall or heart failure). Accelerometers [37,23] are a popular type of sensor for activity recognition and to assess physical activity. Other works in this field rely on numerous sensors to enrich observation data, and often depend on prior knowledge about the activities and the environment learned in a supervised manner.

In [46], Lin et al. present an activity recognition approach using a mobile phone. The data from 6 different test subjects is collected on a Nokia N97, and used to build an SVM-classifier. Five types of features are explored in this work, including *mean*, *variance*, *correlation*, *FFT-energy* and *frequency-domain entropy*. One

[1] http://www.iot-butler.eu/

of the main factors that can influence the recognition rate is the position where a user is carrying his device. This position can either be in the pocket near the hip, in the front pocket or just in his hand. The influence of this position on the accuracy of predictions is researched in this work.

In [20], Khan et al. focus on the fact that the activity recognition approach should work in real-time. The authors claim that frequency domain features work best, but that these require too much computation to be feasible in a real-time scenario. Benchmark testing is carried out with one specific accelerometer. While not further elaborated on, the authors note that the specific set of features used makes the approach more person dependent. In their multiple-subject scenario, data from multiple test subjects is used to train a classifier. They use this input from multiple subjects irrespective of the physique of the persons, but do note that adding more subjects decreases the performance.

Machine learning techniques are frequently used to model a wide range of human activities and to elicit particular patterns of interest. Some techniques investigate to what extend such models can be learned across different users [7] to address the concern of activity recognition algorithms requiring substantial amounts of labeled training data. However, a detailed discussion on human activity recognition is beyond the scope of this work. We refer interested readers to recent surveys [1,6] for a more elaborate and in-depth overview of human activity recognition research.

2.2. Batch and stream processing on big data

The data explosion of the Internet of Things is often linked with the *Big Data* paradigm. MapReduce [9] – and its Hadoop [48] implementation – is a software framework and programming model that allows developers to write programs that process massive amounts of unstructured data in parallel across a distributed cluster of computers. The shortcomings and drawbacks of batch-oriented data processing have been widely recognized as many applications are in need of real-time [44,5] and in-stream processing capabilities [8]. This concept got a lot of traction with various distributed event stream processing (ESP) engines emerging. Yahoo's S4 [29] and Twitter's Storm project [47,26] were among the first to attract a lot of attention. Also Google acknowledged the limitations of MapReduce, with its MillWheel [2] framework and programming model dedicated to fault-tolerant stream processing at Internet scale. Spark [49] is another state-of-practice software solution for large-scale data processing. It runs up to 100 times faster in memory than Hadoop MapReduce and supports scalable fault-tolerant streaming applications. Spark Streaming [50] builds upon the Spark's foundations to build big data application that act on data in real time. Apache Samza[2] – originally developed at LinkedIn – is another popular open source frameworks for scalable stream processing.

2.3. Scalable machine learning and mining

The growing amounts of data has increased the interest in implementing machine learning algorithms on top of the MapReduce framework [14,15]. Mahout [30], a machine learning and data mining framework built on top of Hadoop, addresses this scalability challenge. MLbase [22] builds upon the Spark framework and more particularly the MLI [43] API and Spark's MLlib[3] scalable machine learning library.

Stream mining [13] differs from more traditional data mining libraries like Weka [17] and statistical learning tools like R [18] in the sense that they focus on extracting knowledge in non-stopping streams of events. Massive Online Analysis (MOA) [21] is such a software framework that offers a variety of algorithms and evaluation methods for supervised and unsupervised learning, supporting only single machine deployments thereby limiting its scalability. SAMOA [28] on the other hand is pluggable architecture that allows it to run on several distributed stream processing engines such as Storm [47], S4 [29], and Samza.

3. Motivating use cases

In this section, we present two different use cases. A first use case deals with activity recognition in the healthcare domain. A second use case focuses on the use of context information to implement an intelligent authentication system.

3.1. Activity recognition for e-health applications

A first use case builds upon our mobile application for diabetes patients [32,33] as depicted in Figure 1. The major challenge in this application is to identify classes of activities that have an effect on blood glu-

[2] http://samza.apache.org/

[3] https://spark.apache.org/mllib/

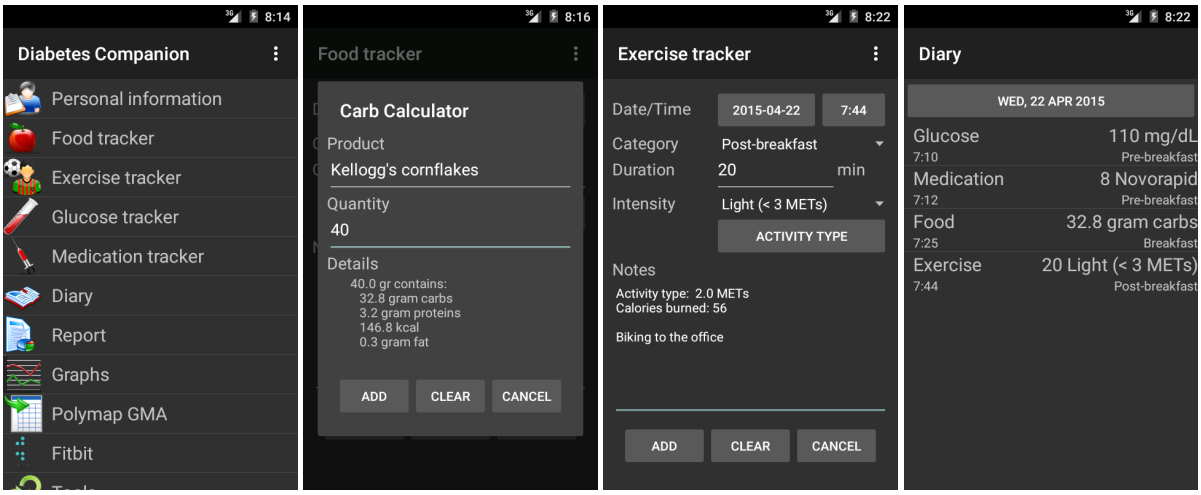

Fig. 1. A mobile context-aware diabetes application as a first motivating use case

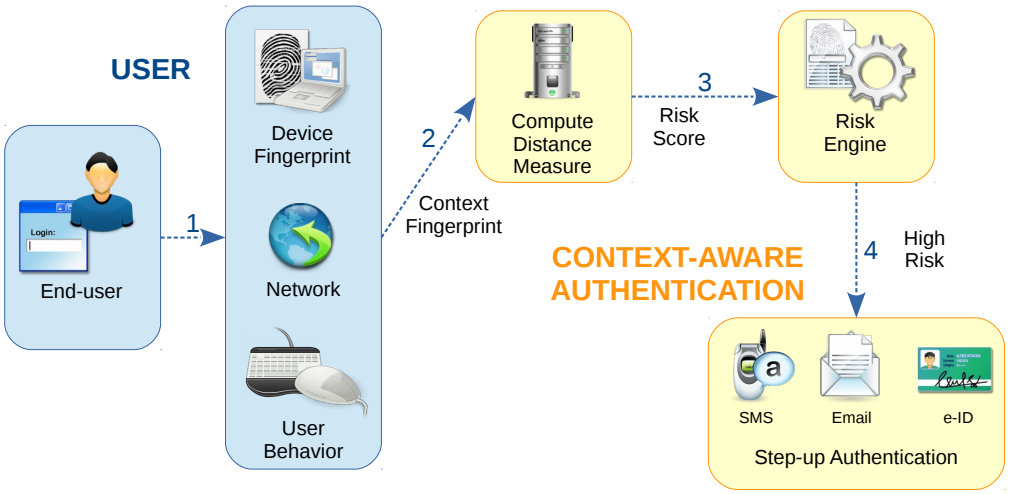

Fig. 2. User-friendly context-aware authentication as a second motivating use case

cose levels. As *activities of daily living* (ADL) typically present recurring behavioral patterns, we explore correlations between *time* and *location* on the one hand and types of activities on the other hand, to find similar situations of the past as a recommendation for the patient. We also track the number of steps taken each day as a measure for well-being and as a means for recommendation to have a more active lifestyle.

3.2. *User-friendly context-aware authentication*

Traditional systems for identity and access management technologies rely heavily on usernames and passwords for authentication. However, weak passwords do not offer the security guarantees for risk-sensitive services that require stronger continuous identity assurance. Furthermore, entering long and hard-to-remember passwords is deemed inconvenient for mobile customers, especially for online applications where a frictionless experience is paramount. This motivating use case originates from our earlier work [36] that investigates non-intrusive authentication techniques that can operate silently in the background based on additional context and behavioral information [41]. Continuous passive assessment of the context, as depicted in Figure 2, enables service providers to streamline access for trusted combinations of user accounts and contexts. Related work [10] has

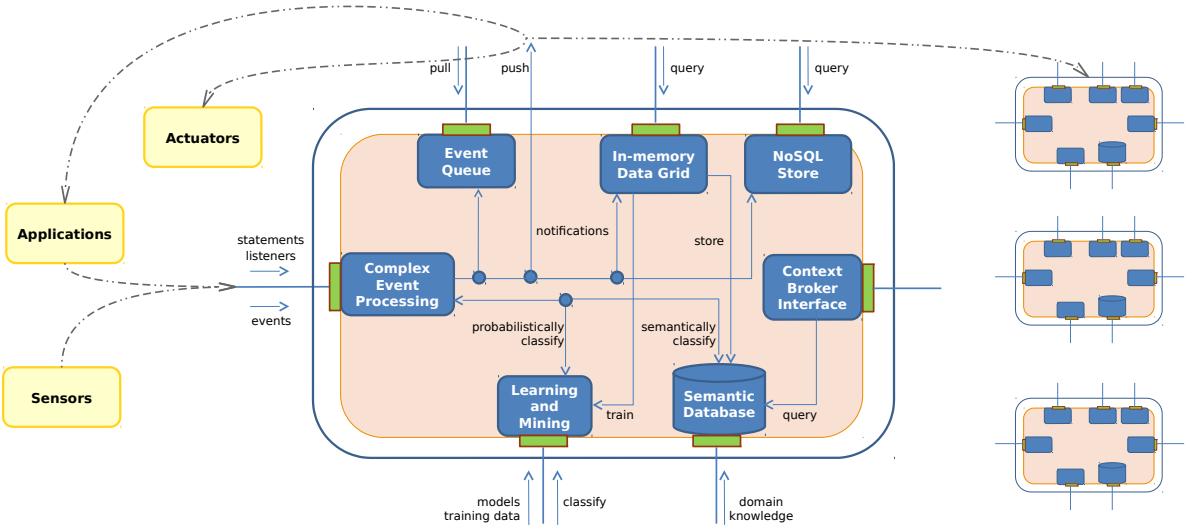

Fig. 3. Early version of the SAMURAI streaming context architecture

shown that particular imperfections of an accelerometer can be used to track a smartphone. We are not pursuing this technique to explore privacy concerns, but rather as a means to recognize trusted devices.

3.3. Common characteristics

Both motivating use cases rely at least in part on accelerometer data and behavior recognition. By adding complexity, such as machine learning and semantic reasoning techniques, we can further improve the recognition accuracy:

- **Feature extraction:** Convert raw sensor data into meaningful features (e.g. walking, running, number of steps, activity intensity)
- **Information fusion:** Aggregate data from different sources to increase the confidence in the quality of the inferred information (e.g. current activity w.r.t. to current time and location)
- **Domain knowledge:** Leverage background information to narrow down likely activities (e.g. semantically linking locations with activity types)
- **Probabilistic correlations:** Identify frequent co-occurrences in event streams to derive event patterns of interest (e.g. spatio-temporal patterns)

However, such complexity is usually too much to handle for a smartphone. Even for server-side implementations scalability remains a concern when multiple customers with different needs must be served at the same time. In the following section, we will discuss the basic primitives that we use in our framework to address these challenges.

4. Batch and streaming context management

The earlier version of SAMURAI – as discussed in [35] and depicted in Figure 3 – was mainly focused on stream-based processing of context information. The old design offered publish/subscribe capabilities to have clients (applications or subsystems) notified when particular (patterns of) events occur with *push* notifications implemented as REST callbacks (see following section). The architecture had three basic components to hold events:

- **In-memory Data Grid**: This is a distributed in-memory container for events based on Hazelcast[4].
- **Event Queue**: Clients that do not support push notifications through REST callbacks can register a queue to hold events and poll that instead.
- **NoSQL Store**: The events can be optionally stored in a persistent way using CouchDB[5] as a RESTful database.

Their RESTful APIs followed the CRUD mapping on HTTP methods to create (POST), read (GET), update (PUT) or delete (DELETE) events.

[4]http://www.hazelcast.com/
[5]http://couchdb.apache.org/

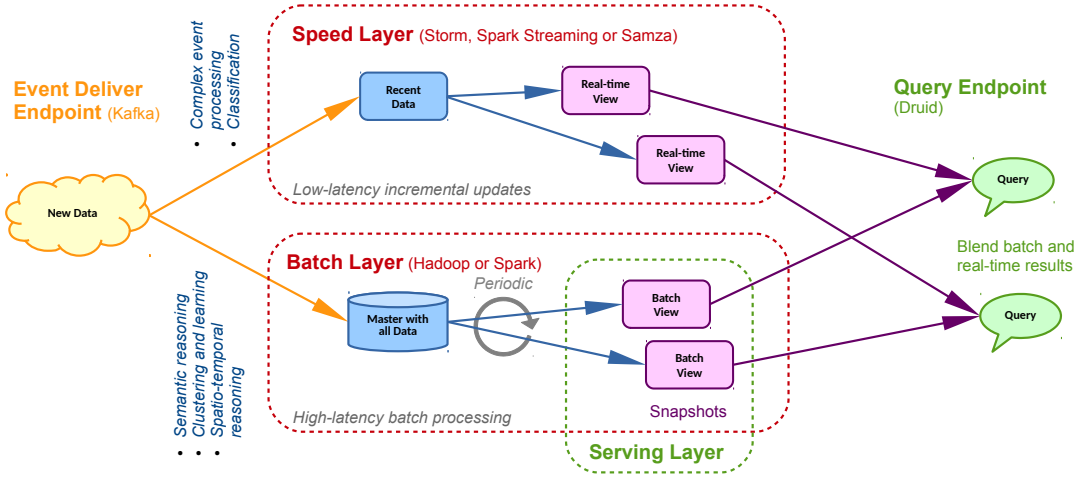

Fig. 4. Conceptual and technology-agnostic overview of a typical Lambda architecture

4.1. Motivations for redesigning SAMURAI

SAMURAI pursued simple methods for scaling out over multiple nodes by replicating functionality and building blocks on different machines. However, the event processing building blocks were not adequate enough to handle large amounts of data effectively. Here are some high-level reasons for why we decided to redesign our SAMURAI system:

1. For more effective context recognition, we wanted to train and test with specialized machine learning algorithms that would operate on all the available data. The previous design and deployment configuration could no longer handle all data effectively with Weka running on a single node, nor was it feasible to carry this out in a streaming fashion. We needed better support for batch-based distributed context processing.
2. For applications that required large amounts of data to be processed, the simple sharding technique we used to scale out resulted into performance problems, mainly due to network latency and bandwidth issues. We needed to take *data locality* into consideration and move the computation where the data is in order to avoid as many data shuffles over the network as possible.
3. The fairly rigid distributed deployment scheme of and coordination between multiple SAMURAI instances (as depicted in Figure 3) was fine-tuned for a particular runtime scenario, but the deployment and configuration became less effective when the amount or the speed of information evolved. Furthermore, our solution was not able to deal with stragglers, i.e. nodes that would take an unusually long time to complete a task, potentially degrading the overall performance.
4. Fault tolerance becomes a more critical concern for large-scale deployments. Big data systems are designed with failure as the norm, rather than as the exception, because hardware failures or network partitions could otherwise lead to unexpected data losses. Our old solution did not have any adequate means to gracefully handle such concerns. It did incorporate data replication to account for nodes being disconnected, but events in transit could get lost and as a result not be processed correctly.

By redesigning SAMURAI, we now aim to fill this gap by offering a distributed and multi-tenant event-based *batch* and *streaming* context architecture with complex event processing, machine learning and semantic context enrichment as key capabilities.

Many of the large-scale distributed data processing systems discussed earlier (e.g. Hadoop, Spark, Storm) have shown their merit in the enterprise for business analytics applications to deal with the above non-functional concerns. In this work we explore the feasibility of such systems for large-scale context-aware applications.

4.2. Basic principles of the Lambda architecture

We redesigned our solution around key concepts of the *Lambda Architecture* – a term coined by Nathan Marz [27] – to achieve processing and serving of extremely high volumes of data in an efficient, scal-

Listing 1 Example of an event type and instance

```
// Event type
{
  "timestamp": "long",
  "x":         "double",
  "y":         "double",
  "z":         "double"
}


// Event instance
{
   "timestamp": 1340099550210,
   "x":         -8.308,
   "y":         -1.9477,
   "z":         4.099
}
```

able and fault-tolerant manner, while maintaining a responsive interaction with the user. Conceptually, the Lambda Architecture – as depicted in Figure 4 consists of three layers: (1) the *Batch* layer which ingests and stores large amounts of *historical* data, and computes views from that data; (2) the *Speed* layer which ingests and processes incremental updates on that data in a low-latency streaming manner, and (3) the *Serving* layer that exposes precomputed views to serve ad-hoc queries with low latency.

Any new data is stored in the batch layer, but also sent to the speed layer. The data in the batch layer is ingested using periodic bulk updates with map-reduce operations that work on the entire master data set (e.g. an immutable store in HDFS). Computations in the batch layer are high-latency and may take hours to complete, and the results of these recomputations are the batch views. After each recomputation, the existing batch views are swapped with the new ones. The speed layer only deals with new data and produces real-time views that compensate for the high-latency updates of the serving layer. The incremental updates in the speed layer are low-latency and usually happen in the order of seconds. The final results for a query merges the output of both the batch and real-time views.

4.3. Basic context event types

Events can be *simple events* that carry slivers of meaning in themselves, and *complex events* which summarize, represent, or denote a set of single events which combined denotes a 'pattern of events'. An event is represented as a set of typed key-value pairs that can be easily serialized into the JSON format. The example in Listing 1 illustrates the type and an instance of an accelerometer event that we use for activity recognition.

In this example, the x, y and z values hold the acceleration values along these axes. The *timestamp* field represents the number of milliseconds passed since January 1, 1970 UTC.

4.4. Batch and stream computation building blocks

The Lambda Architecture is technology agnostic. For each of the various stages in the data pipeline and layers, we use Apache Kafka[6] as a high-throughput distributed publish/subscribe data transport mechanism, Apache Spark[7] for large-scale data processing in the batch layer and the speed layer. For the speed layer, we can alternative choose Apache Storm[8]. Contrary to the Spark Streaming extension that processes micro-batches in a streaming fashion, does Storm (and its Spouts and Bolts) allow for individual event processing for even lower latency stream processing. The batch layer reads all historic input from HDFS, and stores the batch views in HDFS. For the query layer, we offer RESTful services but can also leverage Kafka to publish updates, or use Druid[9] for interactive analytics at scale on large sets of seldom-changing data.

4.5. Distributed semantic reasoning

We use Apache Spark in the batch layer of SAMURAI to implement distributed algorithms that have to process large volumes of data in a scalable way. One of these algorithms is semantic reasoning. For example, we have implemented a distributed RDFS engine with other extensions that reasons on RDF triples based on the following inference rules:

- xxx rdfs:subPropertyOf yyy ∧ yyy rdfs:subPropertyOf zzz ⇒ xxx rdfs:subPropertyOf zzz
- xxx rdfs:subClassOf yyy ∧ yyy rdfs:subClassOf zzz ⇒ xxx rdfs:subClassOf zzz
- aaa rdfs:subPropertyOf bbb ∧ xxx aaa yyy ⇒ xxx bbb yyy
- aaa rdfs:domain xxx ∧ yyy aaa zzz ⇒ yyy rdf:type xxx
- aaa rdfs:range xxx ∧ yyy aaa zzz ⇒ zzz rdf:type xxx
- xxx rdfs:subClassOf yyy ∧ zzz rdf:type xxx ⇒ zzz rdf:type yyy
- xxx rdf:type rdfs:ContainerMembershipProperty ⇒ xxx rdfs:subPropertyOf rdfs:member
- xxx rdf:type rdfs:Datatype ⇒ xxx rdfs:subClassOf rdfs:Literal

[6] http://kafka.apache.org/
[7] http://spark.apache.org/
[8] http://storm.apache.org/
[9] http://druid.io/

Listing 2 Distributed semantic reasoning with RDF triples on top of Apache Spark

```
SparkConf sparkConf = new SparkConf().setAppName("SemanticReasoner");
JavaSparkContext sc = new JavaSparkContext(sparkConf);
JavaRDD<String> lines = sc.textFile(args[0]);
JavaRDD<Triple> triples = lines.map(e -> new Triple(e));

// xxx subPropertyOf yyy . yyy subPropertyOf zzz => xxx subPropertyOf zzz
// xxx subClassOf yyy . yyy subClassOf zzz => xxx subClassOf zzz
{
  JavaRDD<Triple> result = triples.flatMap(a -> {
    List<Triple> list = new LinkedList<Triple>();
    if (a.predicate.equals(Types.RDFSSUBCLASSOF)) {
      List<String> superclasses = getRecursive(a.object, bcSubClass.value());
      for (String superclass : superclasses) {
        list.add(new Triple(a.subject, Types.RDFSSUBCLASSOF, superclass));
      }
    } else if (a.predicate.equals(Types.RDFSSUBPROPERTYOF)) {
      List<String> superproperties = getRecursive(a.object, bcSubProperty.value());
      for (String superproperty : superproperties) {
        list.add(new Triple(a.subject, Types.RDFSSUBPROPERTYOF, superproperty));
      }
    }
    return list;
  });

  triples = triples.union(result);
  triples = triples.distinct();
}
```

Listing 3 Registering event types (lines 1-2); submit events (lines 4-5); register event statements (lines 7-11) and event listeners (lines 13-14)

```
curl -X POST -d '{ "timestamp": "long", "x": "double", "y": "double", "z": "double" }'
  "http://localhost/samurai/rest/esper/eventtypes/AccelerometerEvent"

curl -X POST -d '{ "type": "AccelerometerEvent", "timestamp": 1234, "x": 5.0, "y": 1.3, "z": 2.1 }'
  "http://localhost/samurai/rest/esper/event"

curl -X POST -d '{ "rule":"insert into MagnitudeEvent(timestamp, magnitude) select timestamp, Math.sqrt(x*x + y*y + z*z)
  as magnitude from AccelerometerEvent" }' "http://localhost/samurai/rest/esper/statements/magnitude"

curl -X POST -d '{ "rule":"insert into MovingAverageEvent(timestamp, movingaverage) select timestamp, avg(magnitude)
  as movingaverage from MagnitudeEvent.win:length(10)" }' "http://localhost/samurai/rest/esper/statements/movingaverage"

curl -X POST -d '{"url":"http://otherhost/myapp/steps/offer"}'
  "http://localhost/samurai/rest/esper/statements/steps/listener"
```

The description logic behind an RDFS reasoner is not as expressive as the language supported by many state-of-practice OWL2 ontology reasoners like Pellet [42] or HermiT [16], but our Spark-based implementation is much more capable of handling large amounts of data by transparently distributing the above inference rules and RDF triples over multiple worker nodes. Listing 2 shows how some of the above rules have been implemented in Spark.

4.6. Complex event processing

When Storm is used in the speed layer, then SAMURAI can embed Esper[10] for on-the-fly processing of complex event streams, as Storm's tuples correspond quite well with Esper's event types.

Esper enables feature extraction from low-level events (e.g. from accelerometer to steps) that would otherwise require manual map-reduce implementations. Esper usually relies on Java POJOs to represent events at compile time. However, in SAMURAI new event types can be created anytime. We therefore expose a RESTful API to dynamically register new event types at runtime. Listing 3 illustrates how to do this with *curl*, a command-line utility commonly found on Linux systems to transfer data from or to a server. Registering the other event types and sending events can be done in a similar way as shown in the same figure.

Below is a short overview of some of the event stream processing steps for our motivating use cases:

[10] http://esper.codehaus.org

Listing 4 Semantic representation of rooms and activities in an apartment

```
ex:Room                 a                   owl:Class;
                        rdfs:subClassOf     geo:Feature .

ex:LivingRoom           a                   ex:Room;
                        rdfs:label          "Living Room";
                        geo:hasGeometry     ex:GeoLivingRoom .

ex:GeoLivingRoom        a                   sf:Polygon;
                        geo:asWKT           "POLYGON ((0.00 9.44,3.80 9.44, 3.80 8.13,8.00 8.13,8.00 13.90,
                                            0.00 13.90,0.00 9.44))"^^sf:wktLiteral .

ex:activity             a                   owl:DatatypeProperty;
                        rdfs:domain         ex:Room;
                        rdfs:range          xsd:string .


ex:LivingRoom           ex:activity         "Watch TV" .
ex:LivingRoom           ex:activity         "Listen to music" .
ex:LivingRoom           ex:activity         "Play game" .
ex:LivingRoom           ex:activity         "Read newspaper" .
```

- **Accelerometer**: It produces a continuous stream of X, Y, Z acceleration data at a certain rate (e.g. 100Hz).
- **Low-pass filter**: A 'moving average' removes high-frequency noise to track, for example, steps as a particular peak pattern.
- **Magnitude filter**: Signal analysis on the magnitude of the acceleration signal is independent of the sensor orientation.
- **Peak filter**: This component extracts maxima and minima in the time domain. A single step is characterized by a particular pattern of these features.
- **High-pass filter**: This component implements a FIR filter to detect sudden and high-frequency changes of the acceleration signal.

Our system offers RESTful APIs to register *statements* and *listeners*. A statement is a continuous query registered with an Esper engine instance that provides results to listeners as new events arrive. In order for applications or subsystems to be notified about the *step* events, we add a listener to this statement as shown in line 13. The example adds a REST callback to *http://otherhost/myapp/steps/offer*, which gets called upon using a HTTP GET request for every step event. The event attributes are appended to the REST callback as url parameters. This way, the *myapp* subscriber is notified about all the event details.

4.7. Semantic and spatio-temporal reasoning

Beyond matching patterns of events and feature extraction, SAMURAI can also leverage background knowledge stored in a semantic database to increase the meaningfulness of an event. Unfortunately, there are currently no mature RDF reasoning engines available that both run on top of a distributed MapReduce-like framework with support for the SPARQL query language and spatio-temporal reasoning. Instead, it uses Parliament 2.7.9[11], a GeoSPARQL enabled storage backend for semantic and spatio-temporal reasoning. For scaling out this component or enforcing isolation per tenant or user, we deploy multiple instances of the engine on different nodes with a load balancer in front to share the workload. The GeoSPARQL instances are not part of the batch/speed layer cluster, but run on dedicated nodes in the same network.

The benefits for adding and integrating semantic and spatio-temporal reasoning capabilities into our SAMURAI system are manifold:

- Describe the spatial characteristics of different locations in your environment (see Listing 4 and Figure 5).
- Use the W3C SSN ontology to describe the sensors and their position
- Translate coordinates into semantic locations (e.g. [6.0, 10.0] being in the *Living Room*)
- Semantically link locations with relevant activities (e.g. *Watch TV* in a *Living Room*)

The following (simplified) statement demonstrates the integration with Esper (see Listing 6):

This statement translates the x and y coordinates (e.g. obtained after signal strength triangulation) of in-

[11] http://parliament.semwebcentral.org/

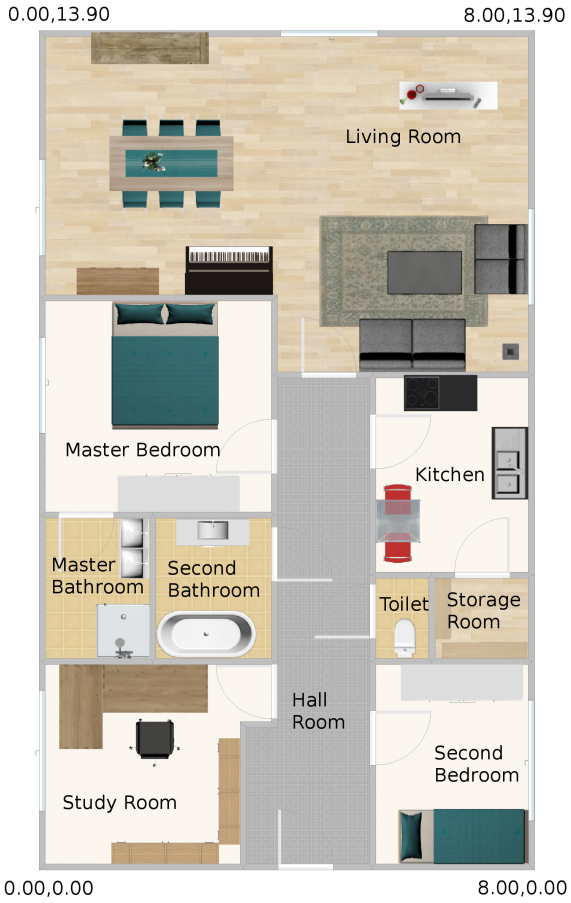

Fig. 5. Visualization of the apartment

coming events of type *LocationEvent* with the custom *location()* Esper operator offered by SAMURAI. The operator is mapped onto a GeoSPARQL query which retrieves the semantic location (e.g. *location(6,10)* → *'Living Room'*). Such higher level concepts are more suitable for classification.

4.8. Learning and mining with classification and clustering

When the relationship between co-occurrent events cannot be established in advance, we need classification and clustering mechanisms to probabilistically infer these dependencies. SAMURAI embeds the Weka [17] machine learning library for this purpose and exposes its key features through RESTful APIs. SAMURAI allows every application to register one or more *models*, with each model having a particular attribute set and classifier. See lines 1 and 2 in Listing 5. This example registers a model called *m01* using Naive Bayes as an incremental classifier. The attributes used for classification are described in the Attribute-Relation File Format (ARFF) and registered with the following REST API (see line 4). By specifying an appropriate statement and corresponding listener, Esper feeds events as training or test instances into the Weka model. The example in lines 6-7 illustrates how to probabilistically classify activities from the current time (in hours) and location (e.g. *8, 'Kitchen'* → *'HavingBreakfast'*). This example demonstrates the use of Weka to learn spatio-temporal correlations. The integration with Esper is again with custom Esper operations mapping the core classification and clustering features of Weka.

Our framework also leverages Spark's MLlib[12] scalable machine learning library, especially for those applications that require processing of vasts amounts of information such that it would become too big to be stored as one Weka model instance.

4.9. Trade-offs and shortcomings

While SAMURAI offers extensive capabilities to process information on a large scale by leveraging state-of-practice building blocks that are exposed as RESTful services, there are some non-trivial trade-offs as SAMURAI offers multiple similar building blocks with different performance capabilities.

- The Weka machine learning library offers more algorithms compared to the MLlib machine learning extension of Apache Spark, but Weka does not have the same distributed processing capabilities for clustering and classification that Spark has.
- A general purpose OWL reasoner like Pellet [42] or HermiT [16] allows for sophisticated semantic reasoning albeit on a single node, whereas our less expressive semantic reasoner can run on Spark in a distributed fashion.
- Complex event processing can be accomplished either leveraging the Esper component or the Spark Streaming framework. Esper offers higher levels of abstraction, whereas the latter is far more easier to scale out over multiple nodes but requires a bit more programming to achieve the same objectives.

One of the shortcomings of SAMURAI is that it does not yet offer a unified abstraction layer for these technologies with common functionalities or similar

[12] https://spark.apache.org/mllib/

Listing 5 Registering a classifier (line 1-2); upload training data (line 4); custom Esper operator for classification (line 6)

```
1 curl -X POST -data '{ "classifier": "weka.classifiers.bayes.NaiveBayesUpdateable" }'
2   "http://localhost/samurai/rest/weka/models/m01"
3
4 curl -X POST --data-binary @m01.arff "http://localhost/samurai/rest/weka/models/m01/arff"
5
6 // Integrating Weka classifier in Esper statement
7 { "rule" : "select time, x, y, classify('m01', hour(time), location(x,y), '?') from ..." }
```

Listing 6 Custom geo-semantic event operator *location()*

```
1 { "rule": "select x,y,location(x,y) from LocationEvent" }
```

algorithms. Such an abstraction layer would trade allow to transparently trade one implementation over the other to address performance concerns.

5. Performance and scalability evaluation

We evaluated our previous version of SAMURAI in [35]. This earlier version also leveraged the key building blocks for complex event processing, machine learning, and spatio-temporal and semantic reasoning. However, the whole architecture was not designed around the Lambda architecture, nor did it incorporate big data processing capabilities. The focus of the evaluation in this work will be on validating the feasibility and performance of our enhanced SAMURAI context architecture, demonstrating near-linear scalability, flexible elasticity and smooth interaction capabilities with the key components of our two motivating use cases.

5.1. Experimental setup

We use an experimental setup of 15 machines, each equipped with an Intel Core 2 Duo 3.00 GHz CPU and 4GB of memory and running a 64-bit Ubuntu 14.04 operating system. All machines are linked to a 1 Gigabit network. We use an additional 5 machines to simulate different users. We refer to the former 15 machines as the *internal* side of the setup, whereas the 5 machines acting as load generators being the *external* side of the experimental setup. Figure 6 illustrates our monitoring dashboard with SAMURAI being deployed on 4 systems (called *laarne*, *ronse*, *temse* and *tremelo*).

From the 15 machines, the front-end is deployed on an Apache Tomcat 8.0.51 application server on a dedicated master node. The semantic reasoning engine of SAMURAI (i.e. Parliament 2.7.8) is deployed on a similar Tomcat instance on two other nodes in a load balanced setup. Parliament is a fully featured RDF triple store that runs on a single machine. To share the workload, the 2 nodes serve the same knowledge graphs so that the GeoSPARQL queries can be distributed and load balanced among both. For our experiments, we do not need more than 2 of such nodes for geospatial and semantic reasoning to not cause any performance bottlenecks, but more Parliament instances can be added if needed.

The 12 other nodes are set up to run the Spark and Storm worker nodes of the batch and speed layers of SAMURAI. They collectively also host the HDFS distributed file system that is used to store all master data processed by the batch layer of SAMURAI.

5.2. Feasibility assessment and validation with motivating use cases

The objective of the motivating examples of section 3 is activity and behavior recognition – common for both use cases – based on accelerometer event streams, spatio-temporal and semantic reasoning. For the first use case, we aim to recognize different types of human motion, whereas in the second use case we continuously analyze and classify the risk for context-aware authentication.

Again, as in the previous work, the objective of the evaluation is not to assess the effectiveness of the recognition, but the scalability of the approach for a large user base. The 5 machines running the load generator that simulates user behavior produces about 50 events per second per user on average as input for SAMURAI. The input constitutes mainly accelerometer and gyroscope events, location updates, wifi network details, and mobile device fingerprints.

Without going into details, these events are processed by batch and speed layers, the complex event processing component, the semantic reasoning engine and machine learning classification algorithms. The batch layer is used periodically offline to train the dif-

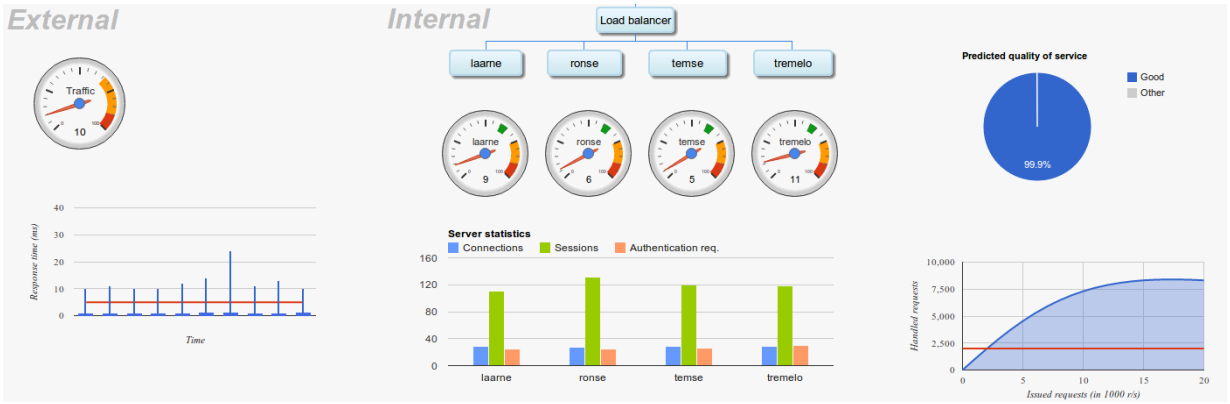

Fig. 6. Monitoring dashboard

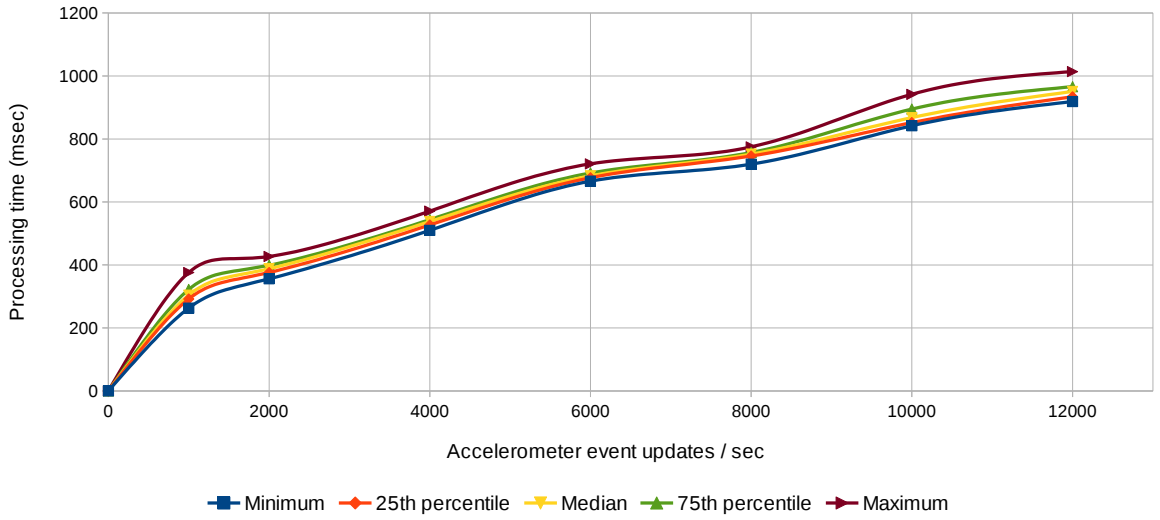

Fig. 7. Event stream processing performance in the speed layer of SAMURAI on a single node

ferent classifiers and decision trees which are used at runtime in the speed layer.

A first experiment particularly focuses on processing the accelerometer events using the Spark Streaming speed layer on a single worker node. The accelerometer runs at about 40Hz for a single user. This means 10000 event updates per second when simulating about 250 users. The results are shown in Figure 7. Simulating more users would cause the accelerometer events to no longer be processed in less than a second, i.e. in a real-time streaming fashion.

For a second experiment, we processing all the events in terms of a growing number of concurrent users in set up scaling out up to 12 worker nodes. Figure 8 illustrates the horizontal scalability.

For both experiments, we assume that the batch layer has already learned the classifiers and decision trees based on all historic data. This step was completed offline in less than 1 hour using all the worker nodes. While our experiments were not carried out on a very large number of nodes (e.g. more than 100), our feasibility assessment does show that SAMURAI exhibits near-linear horizontal scalability. We have identified some performance concerns with Parliament, the GeoSPARQL semantic storage and reasoning backends hosted on the dedicated nodes. As mentioned earlier, Parliament does not support distributed reasoning. We are addressing this concern by implementing distributed RDFS reasoning with *map()* and *reduce()* operations, but it currently does not yet offer the same se-

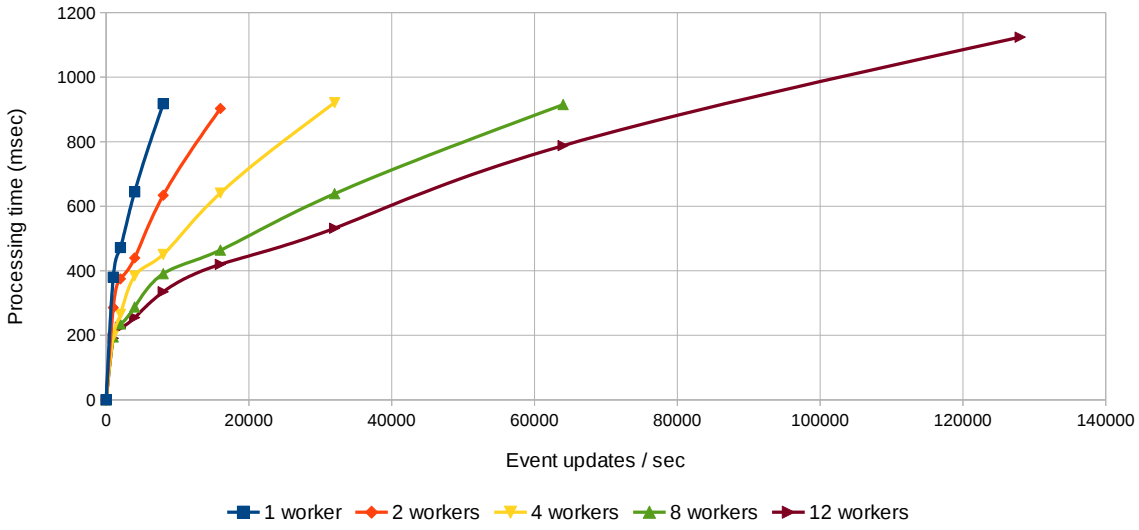

Fig. 8. Scaling out to multiple worker nodes

mantic and spatio-temporal reasoning capabilities that Parliament offers. If such functionality would be available, than all of SAMURAI's features would be running inside the worker node cluster of the batch and speed layers, fully supporting elastic scalability.

5.3. Comparison with state-of-practice solutions

SAMURAI offers a variety of features that are – to the best of our knowledge – not available in any other contemporary framework. To systematically evaluate SAMURAI and compare with systems that share some functionality, there is a need for realistic and commonly accepted benchmarks as well as tool support to generate reproduceable workloads that grow in size and complexity. A side-by-side quantitative comparison of existing systems is therefore not trivial.

However, a quantitative analysis regarding the choice of distributed computing technology for the different layers in SAMURAI is feasible. We tested the batch layer with simple data clustering and outlier detection tasks that were implemented on top of the Hadoop and Spark frameworks. For this particular experiment, Spark far outperformed Hadoop for batch processing tasks with at least an order of magnitude. The main reason for this is that Hadoop saves intermediary processing results on disk whereas Spark is memory oriented. Regarding the speed layer, Apache Spark Streaming cannot achieve the same low latency properties that Storm can. This limits the capabilities of SAMURAI to those use cases that do not require end-to-end data processing latencies below 1 second. However, we believe that these drawbacks do not outweigh the main benefits of having a single Spark API for both the batch and speed layers versus having different implementations for Hadoop and Storm.

6. Conclusion

We presented and evaluated our redesigned version of SAMURAI – our award winning batch and streaming context architecture – that integrates and exposes well-known components for complex event processing (feature extraction, information fusion, notification), machine learning (learn co-occurrences of events and spatio-temporal correlations), and knowledge representation (linking positions with semantic locations and activities).

We redesigned SAMURAI around key concepts of the Lambda architecture – with a batch, speed and service layer – and leveraged big data enabling technologies to achieve horizontal scalability and responsive interaction with its users. Key reasons for the redesign were (1) better support for distributed batch processing on large amounts of data, especially for machine learning tasks, (2) avoid network capacity concerns by taking data locality into consideration, (3) enable dynamic scheduling of distributed tasks to handle stragglers, and (4) consider failure as the norm rather than as the exception by leveraging fault tolerance capabilities of big data processing subsystems.

Two application cases in the healthcare domain and context-aware authentication were used to validate the feasibility and performance of our redesigned SAMURAI context architecture. The experimental evaluation demonstrated near-linear scalability, though a current limitation is still the fact that the semantic and spatio-temporal reasoner is not fully distributed on top of the big data frameworks. In our current setup, we replicate multiple instances of this component in a load balanced configuration to distribute the workload, but this is less effective compared to being able to execute such tasks on multiple nodes in parallel.

In our evaluation, we used the Apache Spark framework and its Streaming and MLlib extensions to implement the batch and speed layers. There are preliminary integrations of other big data frameworks in SAMURAI, especially Apache Storm for the speed layer. However, a side-by-side performance comparison has not yet been carried out because the current implementation of the motivating use cases is tied too much to the underlying big data processing technology. For example, a new implementation of the use cases would be required to make use of the Storm streaming backend in the speed layer. As part of future work, we will explore to what extend we can unify the implementation in a similar way as our Spark prototype where reuse of application code across the batch and speed layer is much more straightforward because these layers rely on the same technology and similar APIs.

Also as future work, we will evaluate SAMURAI's support for fault tolerance and quantify the impact of stragglers. One of the reasons we did not explore this in depth at this stage is because the big data software systems we integrated have different strategies for fault tolerance. Some rely on data replication, whereas others rely on snapshots and the lineage of the data life cycle to recompute lost data or partitions. A systematic comparison would require an adequate testing framework that can introduce realistic faults and performance bottlenecks into the distributed system, both at the network level as well as in the worker nodes.

Acknowledgments

This research is partially funded by the Research Fund KU Leuven.

References

[1] J. Aggarwal and M. Ryoo, Human activity analysis: A review, *ACM Comput. Surv.*, **43**(3):16:1–16:43, April 2011.

[2] T. Akidau, A. Balikov, K. Bekiroglu, S. Chernyak, J. Haberman, R. Lax, S. McVeety, D. Mills, P. Nordstrom, and S. Whittle, MillWheel: Fault-Tolerant Stream Processing at Internet Scale, In *Very Large Data Bases*, pages 734–746, 2013.

[3] M. Arnhold, M. Quade, and W. Kirch, Mobile applications for diabetics: A systematic review and expert-based usability evaluation considering the special requirements of diabetes patients age 50 years or older, *J Med Internet Res*, **16**(4):e104, Apr 2014.

[4] L. Atzori, A. Iera, and G. Morabito, The Internet of Things: A survey, *Comput. Netw.*, **54**(15):2787–2805, October 2010.

[5] M. Barlow, Real-Time Big Data Analytics: Emerging Architecture, Technical report, O'Reilly, June 2013.

[6] A. Bulling, U. Blanke, and B. Schiele, A tutorial on human activity recognition using body-worn inertial sensors, *ACM Comput. Surv.*, **46**(3):33:1–33:33, January 2014.

[7] D. Cook, K. Feuz, and N. Krishnan, Transfer learning for activity recognition: a survey, *Knowledge and Information Systems*, **36**(3):537–556, 2013.

[8] G. Cugola and A. Margara, Processing Flows of Information: From Data Stream to Complex Event Processing, *ACM Comput. Surv.*, **44**(3):15:1–15:62, June 2012.

[9] J. Dean and S. Ghemawat, MapReduce: simplified data processing on large clusters, *Commun. ACM*, **51**(1):107–113, January 2008.

[10] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, Accelprint: Imperfections of accelerometers make smartphones trackable, In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*, The Internet Society, 2014.

[11] B. DM, S.-S. C, S. V, and et al, Using pedometers to increase physical activity and improve health: A systematic review, *JAMA*, **298**(19):2296–2304, 2007.

[12] A. Dohr, R. Modre-Opsrian, M. Drobics, D. Hayn, and G. Schreier, The Internet of Things for Ambient Assisted Living, In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 804–809, 2010.

[13] M. Gaber, A. Zaslavsky, and S. Krishnaswamy, Mining data streams: a review, *SIGMOD Rec.*, **34**(2):18–26, June 2005.

[14] A. Ghoting, P. Kambadur, E. Pednault, and R. Kannan, Nimble: A toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce, In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 334–342, New York, NY, USA, 2011, ACM.

[15] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan, Systemml: Declarative machine learning on mapreduce, In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 231–242, Washington, DC, USA, 2011, IEEE Computer Society.

[16] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, Hermit: An owl 2 reasoner, *Journal of Automated Reasoning*, **53**(3):245–269, 2014.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, The WEKA data mining software: an update, *SIGKDD Explor. Newsl.*, **11**(1):10–18, November 2009.

[18] K. Hornik, C. Buchta, and A. Zeileis, Open-source machine learning: R meets weka, *Computational Statistics*, **24**(2):225–232, 2009.

[19] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, Big data and its technical challenges, *Commun. ACM*, **57**(7):86–94, July 2014.

[20] M. Khan, S. I. Ahamed, M. Rahman, and R. O. Smith, A Feature Extraction Method for Real time Human Activity Recognition on Cell Phones, In *isQoLT 2011*.

[21] P. Kranen, H. Kremer, T. Jansen, T. Seidl, A. Bifet, G. Holmes, B. Pfahringer, and J. Read, Stream data mining using the moa framework, In *Database Systems for Advanced Applications*, volume 7239 of *Lecture Notes in Computer Science*, pages 309–313, Springer Berlin Heidelberg, 2012.

[22] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, Mlbase: A distributed machine-learning system, In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, www.cidrdb.org, 2013.

[23] J. Kwapisz, G. Weiss, and S. Moore, Activity recognition using cell phone accelerometers, *SIGKDD Explor. Newsl.*, **12**(2):74–82, March 2011.

[24] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, A survey of mobile phone sensing, *Comm. Mag.*, **48**(9):140–150, September 2010.

[25] H. Lee, Y. S. Choi, and S. Lee, Mobile posture monitoring system to prevent physical health risk of smartphone users, In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 592–593, New York, NY, USA, 2012, ACM.

[26] J. Leibiusky, G. Eisbruch, and D. Simonassi, *Getting Started with Storm - Continuous Streaming Computation with Twitter's Cluster Technology*, O'Reilly, 2012.

[27] N. Marz and J. Warren, *Big Data. Principles and best practices of scalable realtime data systems*, Manning Publications Co., April 2015.

[28] G. D. F. Morales and A. Bifet, Samoa: Scalable advanced massive online analysis, *Journal of Machine Learning Research*, **16**:149–153, 2015.

[29] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, S4: Distributed Stream Computing Platform, In *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops*, ICDMW '10, pages 170–177, Washington, DC, USA, 2010.

[30] S. Owen, R. Anil, T. Dunning, and E. Friedman, *Mahout in Action*, Manning Publications Co., Greenwich, CT, USA, 2011.

[31] E. Ozdalga, A. Ozdalga, and N. Ahuja, The smartphone in medicine: A review of current and potential use among physicians and students, *J Med Internet Res*, **14**(5):e128, Sep 2012.

[32] D. Preuveneers and Y. Berbers, Mobile phones assisting with health self-care: a diabetes case study, In *Mobile HCI*, ACM International Conference Proceeding Series, pages 177–186, ACM, 2008.

[33] D. Preuveneers, Y. Berbers, and W. Joosen, The future of mobile e-health application development: exploring HTML5 for a context-aware diabetes monitoring assistant, In *3rd International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare*, October 2013.

[34] D. Preuveneers and Y. Berbers, Mobile phones assisting with health self-care: A diabetes case study, In *Proceedings of the 10th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '08, pages 177–186, New York, NY, USA, 2008, ACM.

[35] D. Preuveneers and Y. Berbers, SAMURAI: A streaming multi-tenant context-management architecture for intelligent and scalable internet of things applications, In *2014 International Conference on Intelligent Environments, Shanghai, China, June 30 - July 4, 2014*, pages 226–233, IEEE, 2014.

[36] D. Preuveneers and W. Joosen, Smartauth: Dynamic context fingerprinting for continuous user authentication, In *Proceedings of the 30th ACM/SIGAPP Symposium on Applied Computing (SAC 2015), Salamanca, Spain, April 13-17, 2015*, 2015.

[37] N. Ravi, N. Dandekar, P. Mysore, and M. Littman, Activity recognition from accelerometer data, In *Proceedings of the 17th conference on Innovative applications of artificial intelligence - Volume 3*, IAAI'05, pages 1541–1546, 2005.

[38] R. Rawassizadeh, B. A. Price, and M. Petre, Wearables: Has the age of smartwatches finally arrived?, *Commun. ACM*, **58**(1):45–47, December 2014.

[39] M. J. Rotheram-Borus, M. Tomlinson, D. Swendeman, A. Lee, and E. Jones, Standardized functions for smartphone applications: Examples from maternal and child health, *Int. J. Telemedicine Appl.*, **2012**:21:21–21:21, January 2012.

[40] C. Scanaill, S. Carew, P. Barralon, N. Noury, D. Lyons, and G. Lyons, A review of approaches to mobility telemonitoring of the elderly in their living environment, *Annals of Biomedical Engineering*, **34**(4):547–563, 2006.

[41] E. Shi, Y. Niu, M. Jakobsson, and R. Chow, Implicit authentication through learning user behavior, *Information Security*, pages 99–113, 2011.

[42] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, Pellet: A practical owl-dl reasoner, *Web Semant.*, **5**(2):51–53, June 2007.

[43] E. R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, M. J. Franklin, M. I. Jordan, and T. Kraska, Mli: An api for distributed machine learning, *2013 IEEE 13th International Conference on Data Mining*, **0**:1187–1192, 2013.

[44] M. Stonebraker, U. Cetintemel, and S. Zdonik, The 8 requirements of real-time stream processing, *ACM SIGMOD Record*, **34**(4):42–47, December 2005.

[45] M. Sumida, T. Mizumoto, and K. Yasumoto, Estimating heart rate variation during walking with smartphone, In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 245–254, New York, NY, USA, 2013, ACM.

[46] L. Sun, D. Zhang, B. Li, B. Guo, and S. Li, Activity recognition on an accelerometer embedded mobile phone with varying positions and orientations, In *Proceedings of the 7th international conference on Ubiquitous intelligence and computing*, UIC'10, pages 548–562, Berlin, Heidelberg, 2010, Springer-Verlag.

[47] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, Storm@twitter, In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 147–156, New York, NY, USA, 2014, ACM.

[48] T. White, *Hadoop: The Definitive Guide*, O'Reilly Media, Inc., 1st edition, 2009.

[49] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012, USENIX As-

sociation.

[50] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, Discretized streams: Fault-tolerant streaming computation at scale, In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 423–438, New York, NY, USA, 2013, ACM.

[51] N. Zouba, F. Bremond, and M. Thonnat, Multisensor fusion for monitoring elderly activities at home, In *Advanced Video and Signal Based Surveillance, 2009. AVSS '09. Sixth IEEE International Conference on*, pages 98–103, 2009.