CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Cloudifying a legacy batch-processing system

A Design Science Study

Bachelor of Science Thesis in Software Engineering and Management

ANDREAS ARONSSON
LINHANG NIE

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Cloudifying a legacy batch-processing system

ANDREAS ARONSSON
LINHANG NIE

Supervisor: Imed Hammouda
Supervisor: Michal Palka
Examiner: Hang Yin

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Cloudifying a legacy batch-processing system

Andreas Aronsson
Gothenburg University
Gothenburg, Sweden
gusaroanc@student.gu.se

Linhang Nie
Gothenburg University
Gothenburg, Sweden
guslinhni@student.gu.se

***Abstract*—Cloud services have become increasingly popular in the past years. The attractiveness of reduced operational costs and elasticity to fluctuating loads are prompting more organizations to migrate to the cloud. However, there exists inertia when it comes to migrating business critical systems. Often these legacy systems have been under development for years and in many cases decades, which complicates the process. This paper aims to investigate how to cloudify a legacy batch-processing system. Subsequently, cross-cutting concerns of the cloud migration are presented to be used as a foundation for organizations deciding whether to migrate to the cloud.**

***Keywords*-Cloud computing, cloud migration, software migration, legacy-to-cloud migration**

I. Introduction

Legacy systems are core parts of many organizations and with the increase in popularity of cloud computing, benefits such as reduced cost, greater flexibility and efficiency are tempting organizations to make the move to the cloud. These systems, often built with now outdated technology, tend to be hard to evolve and grow [1]. Moreover, they become difficult to maintain as time passes [2].

The migration of legacy systems is a difficult process because of the inherent complexity of the systems. The lack of architectural adaptation guidance and migration plan [3] further complicates the migration process.

A. Background and Problem domain

This thesis project is part of the study of migrating the integrated production system, SPRINT, to the cloud in Volvo Truck and Technology, with a special focus on improving its scalability and performance. The work focuses on the order breakdown system, which is a subsystem of the larger system - SPRINT. The subject system takes in orders with highly customized configurations and outputs all of the material used in the product together with associated assembly instructions. The target system consists of a series of monolithic batch-processing systems that have been under development for the past 15 years. Further descriptions of mentioned concepts follows:

1) Monolithic system: Monolithic systems are software systems that have monolithic architecture, which is a common style used in legacy systems that are deployed on mainframe computers. R. Stephens, [4] describes a monolithic architecture as a single program that does everything. The benefit of a monolithic architecture is that since the system is contained in a single program there is no need to implement and maintain communication protocols. The drawback of this type of architecture is the inflexibility and tight coupling making it difficult to change and adapt to changing business needs such as the need for scalability.

2) Batch processing: Batch processing is a processing mode where individual transactions are grouped or batched and processed periodically as a job [5]. The time it takes to process a job depends on the nature of the application, transaction volume, simultaneous workload and the sequence in which the processing occurs.

The order breakdown system is a good example of a batch processing system since it has the following characteristics:

- The program runs unattended
- Input data is collected over some time
- Output of the system is not needed for use as soon as it is produced

The system has a daily scheduled breakdown batch job to process data. It is an appropriate object to investigate how to migrate a legacy batch-processing system, as it processes large enough amount of data and lack of architectural adaptability, which presents the following issues:

- Low resource utilization. The daily breakdown jobs in each instance only run in a specific time slot each day, the resources are idle the rest of the day.
- Hard to scale-up. The system is hard to scale-up in an optimal way, though it is possible for the current system to partition the data and distribute the jobs among servers, underlying synchronization and consensus risks regarding may occur.
- Fail-over. Whenever a batch job fails the whole batch job has to restart.

The main problems that we are trying to address are the scalability of the batch-processing part and the adaptation of the system to an architecture.

B. Potential solutions and Related Work

In this section, the concepts that are used in the migration process are explained such as Service-Oriented Architecture (SOA) and cloud computing.

1) Service-Oriented Architecture: Service Oriented Architecture, defined by K. Channabasavaiah et al, is an architecture within which functions are defined as services that can be called via well-defined interfaces to form business

processes [6]. It enables the re-engineering and migration of legacy software systems into loosely coupled and interoperable sets of services [7]. S. Alahmari, proposes three different ways of migrating legacy code to SOA [8]. One way is to reverse engineer and to implement the code in another programming language. Another way is by using wrapping and wrap the legacy code and access it through the existing interface. The third way is to transform the code by salvaging the legacy code, wrapping it and making it available as a web service. H.M. Sneed, has presented techniques of migrating COBOL legacy systems to SOA [9] [10]. There have been plenty of research conducted into the domain of SOA migration as reflected in the systematic review by M. Razavian et al [11]. Colosimo et al, presented an experiment with the goal of defining a systematic migration strategy to migrate COBOL systems to the web [12].

2) Cloud computing: As described by M. Armbrust, et al, cloud computing or the cloud refer both to the services delivered over the Internet and the hardware that provides these services [13]. Services belonging under the umbrella term of the cloud are usually referred to as IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service). By utilizing the potential of the cloud, and distributing the software over a pool of resources, organizations can benefit from the scalability, reliability and the "pay-as-you-use" payment model of cloud services.

P. Jamshidi et al [3], identified, classified and compared research on cloud migration. They found that migration research concerning the cloud is still in its early stages of maturity, but is advancing. P. Jamshidi et al, also concluded that there is a lack of tools to automate the cloud migration tasks, and stressed the need for architectural adaptation when undertaking a cloud migration task.

C. Research Goal and Research Questions

This research is conducted with the primary objective to investigate how to cloudify a legacy batch-processing system. More specifically, our vision is to cloudify the batch-processing system using a cloud computing engine and integrate the new system in a way such that it would be able to be used by all interested parties. We aim to answer the following questions:

- RQ1: What are the architectural implications of migrating a legacy batch-processing system to the cloud?

By implication of the migration, we mean two perspectives: the architectural design's perspective and the action's perspective. From architectural design perspectives, we want to know what are the trade-offs related to specific quality attributes, hence we ask the following sub-question:

- RQ1.1: What are the cross-cutting concerns regarding the quality attributes when migrating to the cloud?

From the migration action perspective, we intended to learn the transformation of the architectural elements from the old system to the new system, therefore we ask the following sub-question:

- RQ1.2: What architectural elements have been touched during the migration?

We plan to conduct architectural analysis on the adopted architecture to answer the first sub-question, and compare the architectural elements of the old system and the new system.

D. Contribution

As suggested by Jamshidi et al, the cloud migration research field is still in its early stages of maturity. Among the selected papers reviewed in the systematic literature review, only two focus on architectural adaptations [3]. Besides the architectural adaptation support, challenges and lessons learned in an industrial context are needed as well. Only five out of 23 papers report the experiences and lesson learned. Hence the current body of research knowledge needs architectural adaptation support and experience reports in an industrial context.

This study will propose an architecture to cloudify an industrial legacy batch-processing system and report the architectural adaptations made to the target system to fit into the proposed architecture. This study will report the architectural implications of cloudifying a legacy batch-processing system using cloud computing engine with a special focus on the cross-cutting concerns and transformation of architectural elements.

The concrete architecture, CloudSOA, and its architectural adaptation support, the migration plan, would be the scientific contribution to the current body of literature. Both the architecture and the architectural implications will be useful for further second hand studies. The technical contributions are the proposed architecture, the prototype and the quality trade-offs. These could be used by an organization as a reference to migrate an industrial legacy system to the cloud.

E. Structure

The rest of this paper is structured as follows. Section II outlines the background and problem domain of this paper. Section III describes the methodology, design science, used in this paper. In Section IV the CloudSOA architecture, the prototype and the migration plan is described. Section V presents the evaluation. Section VI discusses the cross-cutting concerns regarding the quality attributes when migrating to the cloud. Subsequently the challenges and lessons learned are discussed. This paper concludes in Section VII.

II. Background and Target System

The target system SPRINT is an integrated production system that takes data from upstream applications and performs computation upon the data and delivers the output to downstream applications. As shown in Fig. 1, the data-processing sub-system consists of three layers: integration layer, application layer and the data layer. The integration layer takes production data from multiple upstream sources and pre-processes the data. One component in the application layer constructs object model using the data and writes the objects

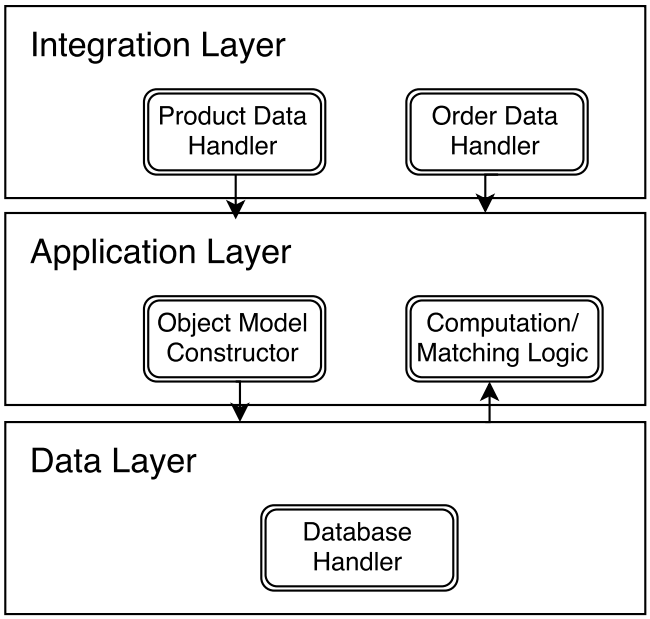

Fig. 1. Current system architecture

to a database via the data layer. Another batch-processing component performs a mass matching job between the processed production data and order data taken from another source, then sends it to other interested parties. There are multiple pairs of SPRINT deployed on a cluster of servers. Each pair has one active instance and one back-up instance for redundancy. From system level, the pairs are used separately to process different batch jobs.

III. Research Methodology

The methodology used in the development of the solution is Design Science, which addresses research through the building and evaluation of artifacts designed to meet identified business needs [14].

In order to answer research question 1, "What are the architectural implications of migrating a legacy batch-processing system to the cloud?", and 1.1, "What are the cross-cutting concerns regarding the quality attributes when migrating to the cloud?", comparisons of the legacy system architecture with the new architecture, referred to as CloudSOA were made.

A non-formal discussion about the current architecture of SPRINT was undertaken with a software architect at the target organization in order to be able to compare the current and the proposed architecture.

An interview based on the ISO/IEC 25010 quality model [15] was conducted in order to get the quality characteristics that are most important to the organization. The quality trade-offs are based on the answers from the interview and related papers, in order to have an informed argument.

A. Artifacts

As seen in Fig. 2, the design science framework, we propose an architecture and developed a prototype. The prototype is an implementation of the order breakdown system that matches restrictions associated with the orders, such as customer configurations, geographical and business restrictions, with the parts or components that go into the final product. Each part, in turn, can have one or many restrictions of their own. The output generated from this matching is the set of parts needed to complete an order.

B. Evaluation

In design science, evaluation is a crucial component of the research process [14]. We will demonstrate the utility of the architecture by validating the prototype against the quality criteria, which are scalability, viability, and performance. For performance, we conduct a controlled experiment measuring the time elapsed per batch of jobs.

The quality criteria mentioned above are derived from the business needs of the stakeholder:

- Scalability - Must be able to easily expand the computing resource pool to meet heavier loads of input.
- Viability - Must meet the stakeholder's demands on infrastructure and compatibility with existing systems.
- Performance - Must meet the stakeholders demands of performance. A batch of jobs must be completed in less than 8 hours.

IV. CloudSOA - A concrete architecture

A. Architecture

1) Idea of CloudSOA: Fig. 3 shows the architecture in different abstraction levels, from reference model to concrete architecture, and the implementation of the architecture. CloudSOA is a concrete architecture that employs cloud computing services in service-oriented architecture paradigm. The goal of the architecture is to make both the data-processing engine and the processing logic reusable for multiple interested parties.

SOA and its essential elements are described in I-B1, the SOA reference model defined the abstract conceptual model of Service-oriented Architecture and its essential elements. MacKenzie, C. Matthew, et al. suggested that Service-Oriented Architecture provides a rather simple paradigm to reconcile Internet-scale provisioning and consumption of capabilities that may be within different ownership domains [16]. A service is an approach to access the capabilities provided by service provider via the prescribed interface. Services behave as described in their specification and whose interfaces should be accessed as the way specified in the service interface specification [16]. Moreover, Service-Oriented Architecture also provides a solid foundation for business agility and adaptability [16]. One of the benefits of adopting Service-Oriented Architecture is its inherent extensibility. Functionality could be added to the system by connecting to the enterprise service bus and providing a general invokable interface to the interested parties.

2) Structure of CloudSOA: The proposed concrete architecture CloudSOA is derived from the SOA reference model with special attention paid to integrating the cloud-computing engine as a service to the enterprise application ecosystem.

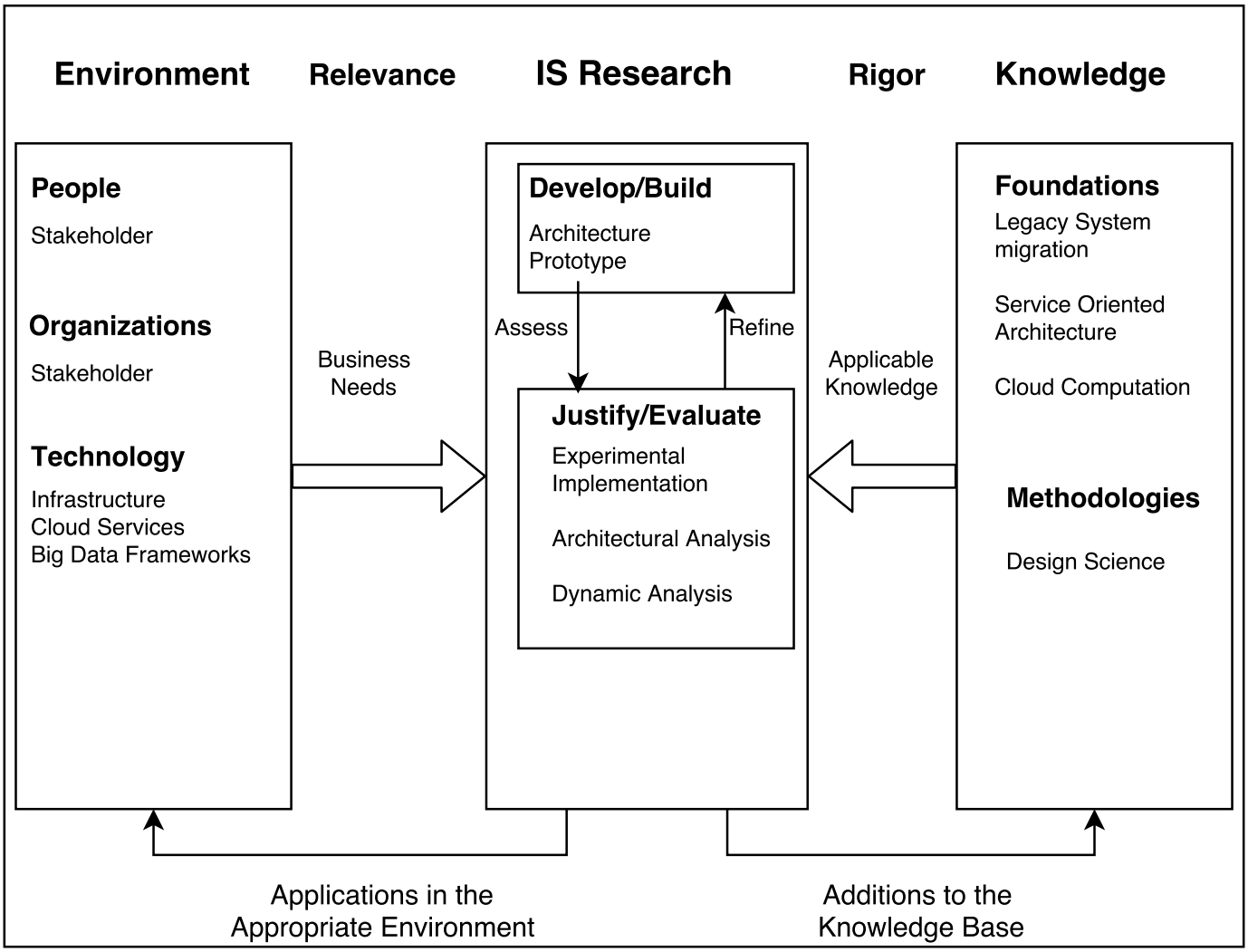

Fig. 2. Design Science Framework adapted from A. Havner et al. [14].

As shown in Fig. 4, the concrete architecture CloudSOA consists of three types of components, which are the service provider, the enterprise service bus, and the service consumers. A service provider in this concrete architecture is a cloud-computing engine wrapped by an interface so that it can be invoked through the service bus by service consumers. The service consumers are other enterprise applications in the ecosystem that need to use the service. Services, service bus and service consumers communicate via protocols of choice.

B. Prototype

In this section, we introduce the implementation of the proposed architecture. As seen in Fig. 5 the computation component of the prototype is implemented using Apache Spark, an engine for large-scale data processing providing APIs in Scala, Python, Java, and R. By using Spark the ability to easily scale the system by adding more nodes is supported out of the box. Another advantage is that Spark is built around the concept of RDDs (Resilient Distributed Datasets) allowing the system to be fault tolerant [17]. For example, if the system is running on 5 nodes and one node goes down, Spark will automatically recompute the missing partitions from the node that went down.

The bus, Kafka, connects the service consumers (order module and product data modules) and the service provider (Spark) as a distributed publish-subscribe messaging system. The prototype also consists of two enterprise applications - the order breakdown module and the product-data update module. The order module takes upstream order data, which can come from one or many different systems and prepares the data for the computation. The product data module takes upstream data and prepares it for further computation. Spark Streaming is an extension to Spark that enables low-latency stream processing. It has the added benefit of allowing queries to be run on arriving streams or historical data using the same API [18].

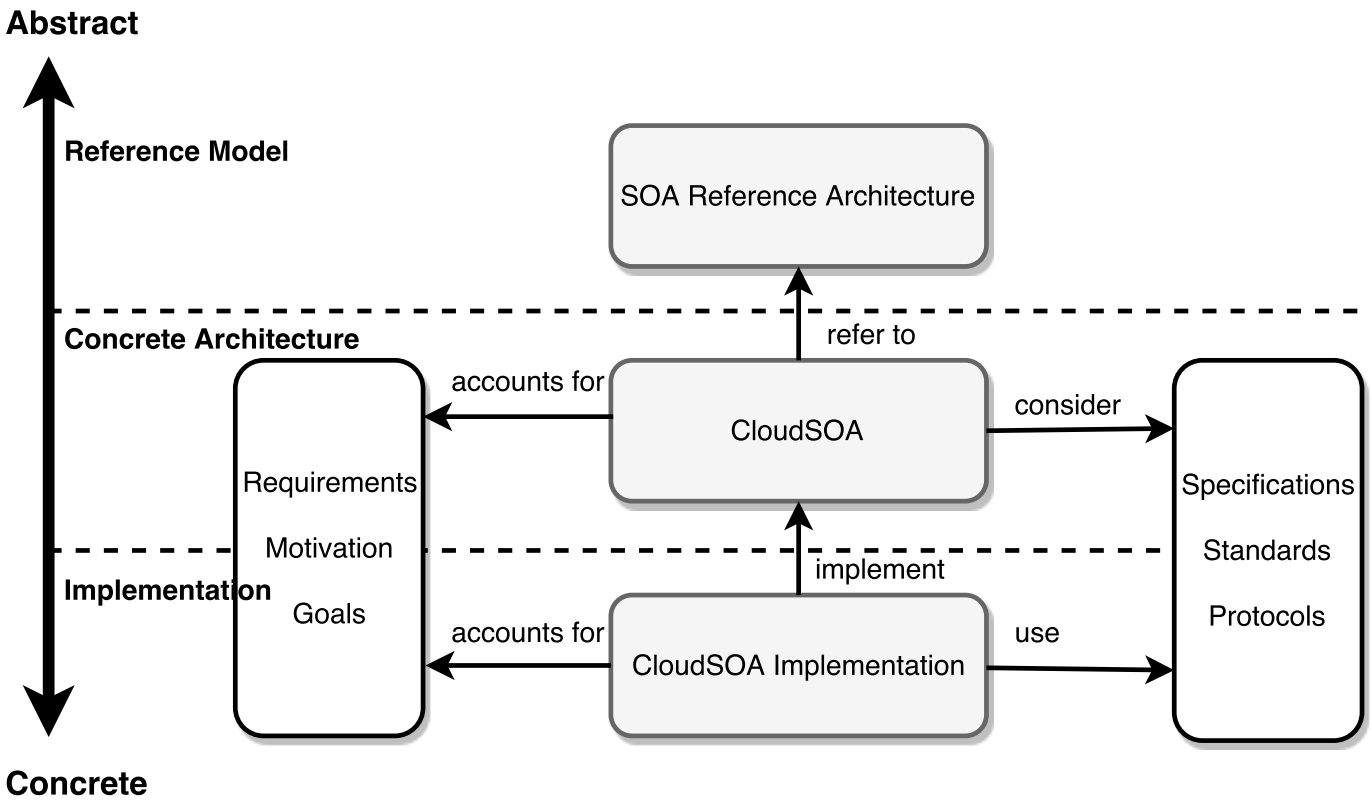

Fig. 3. SOA reference model.

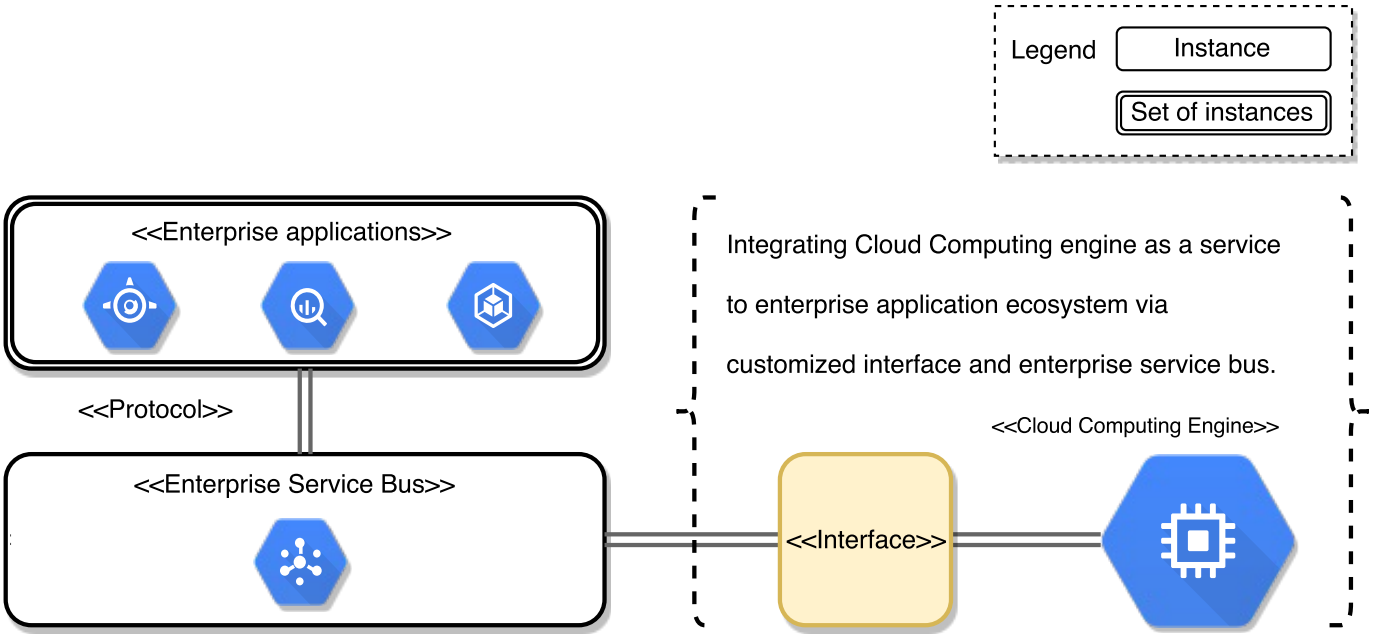

Fig. 4. CloudSOA Architecture.

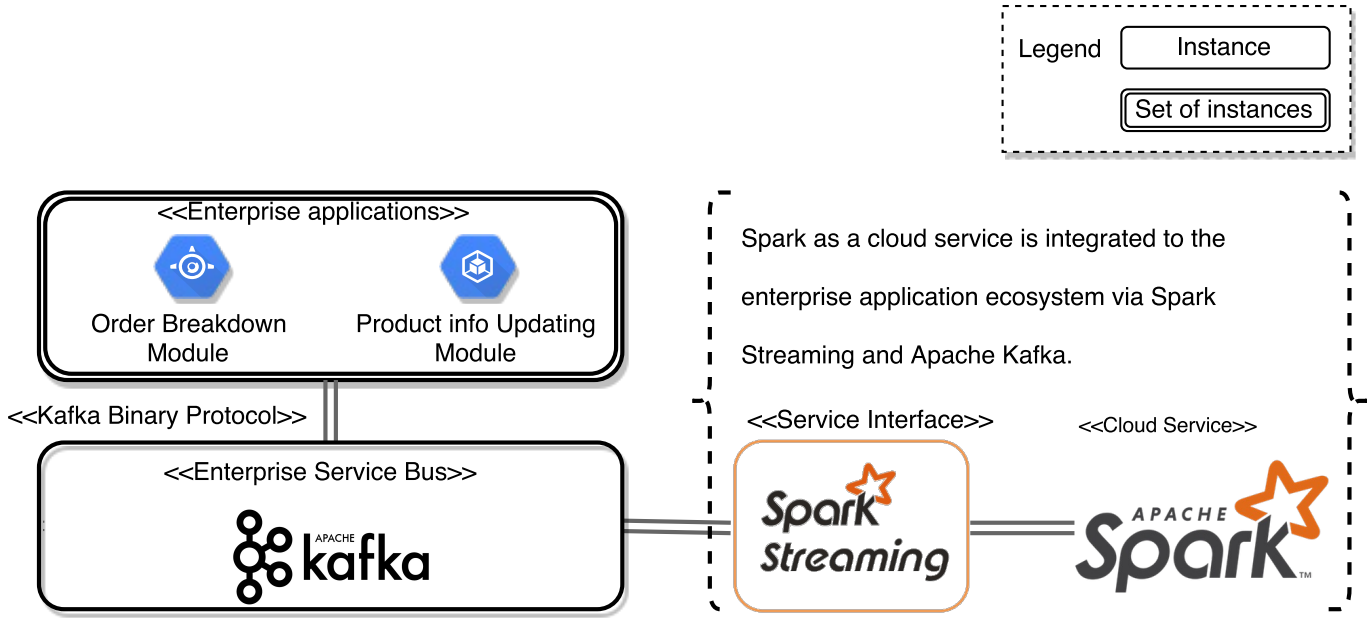

Fig. 5. Architecture of Prototype implementing CloudSOA.

C. Migration plan

This section describes the architectural adaptions made to the order breakdown system in order to fit the CloudSOA architecture. Fig. 6 illustrates the current architecture of the order breakdown system while Fig. 7 shows the proposed architecture. The Integration- and Application-layer of the old architecture are tasked with taking in upstream data and creating data objects in the proper structure. The Breakdown Application layer and Breakdown Data layer are responsible for matching the order data with the product data and outputting the subset of parts from the product data that are valid for an order.

In the proposed architecture this task is being accomplished in the Application layer by two modules, the Order Breakdown module and Product Update module. As the names suggest the Order Breakdown creates the proper data objects for the order data while Product Updater does the same for the product data. The Mediator of the proposed architecture connects the Application layer and Service layer together.

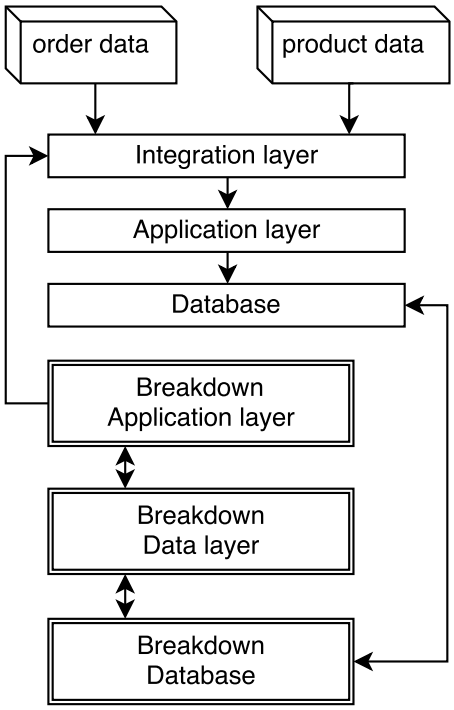

Fig. 6. Order breakdown system current architecture

V. Evaluation

We developed a prototype to evaluate the concrete architecture CloudSOA as it has to be put into a context. Since it was developed in order to meet the identified business needs, CloudSOA has to fulfill the quality criteria of scalability, performance, and viability that are derived from these needs in order to be considered a success.

A. Scalability

Spark is scalable due to the programming model where the programmer creates acyclic data flow graphs to pass input data through a set of operators, allowing Spark to manage scheduling without intervention from the developer [19]. This also works across nodes making it trivial to add another node to the cluster [17].

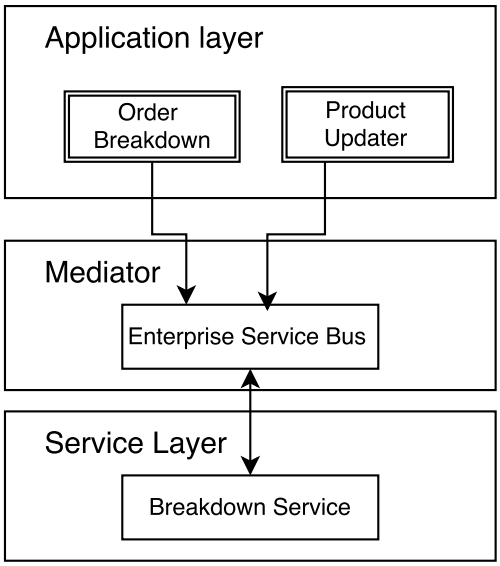

Fig. 7. Order breakdown system proposed architecture

Job amount:	1	10	20
run 1 (s):	19.42	234	450
run 2 (s):	19.36	234.6	451.2
run 3 (s):	19.47	236.4	456
Average time (s):	19.42	235	452.4

TABLE I
RUNTIME IN SECONDS FOR BATCHES OF JOBS

B. Viability

From discussions with the stakeholders, we can conclude that the proposed solution could be viable since there is nothing stopping upstream and downstream components to interface with the proposed solution. However to test it out in practice with the current upstream and downstream components are out of scope for this thesis. There are potential pitfalls and trade-offs that impact the viability that is further discussed in the discussion part of this report.

C. Performance

The results of the throughput test are presented in Table 1. The tests were conducted on a system with Intel i5-3570K CPU @ 3.40GHz and 8GB main memory running the Ubuntu operating system. One job contains one or many orders, each order has around 600 different variants that are matched with product and date restrictions.

VI. DISCUSSION

A. Cross-cutting concerns

While analyzing the concrete architecture and assessing it against the ISO/IEC 25010 quality model, we noticed various architectural implications, such as cross-cutting concerns with regard to security, performance and maintainability.

1) Security: Moving the system to a public cloud could cause some security concerns compared to owning the servers as is the case for the organization today. Since the data is transmitted via the Internet, and the computing service and other related services, such as storage and logging service, are deployed on the cloud infrastructure, the information security is at risk. To mitigate the risk during the transmission, encryption could be taken into consideration. Encrypting sensitive or business critical data, such as pricing information, would ease the security concern for the data transmission process. However, the security of the computing service relies on other security services provided by the cloud service provider. Another option could be deploying the computing service on private cloud in a data center where the infrastructure is owned by the enterprise. As the infrastructure is under the control of the organization, the services deployed on it's private cloud could be protected by customized security systems. Another compromise could be using a hybrid cloud for services with different security levels. Services that dealing with business critical data, such as data storage services, could be deployed on a private cloud for security concerns. And CPU intensive services could be deployed on a public cloud so that they are capable to scale up elastically. Further more, the data sets can be sectioned in order to only expose the computation critical data to the external environment. The outcome could be pieced together in a private cloud afterwards. This approach reconciled the security and the need of scalability, but in turn, brings extra complexity and challenge to data integrity.

2) Availability: Given that the services are deployed on the cloud, the availability of the services rely on the availability of the cloud. While adopting service from the provider, the availability of the cloud should be taken into consideration so that the critical services are available at the expected availability level. Other than the choice of service provider, tactics such as redundancy, which would add extra cost, could also mitigate the availability concern.

3) Performance: Having the system in a public cloud allows the hardware to be more easily scaled compared to owning the hardware. If there is a requirement for more processing power or memory this can instantly be added on the more popular cloud services such as AWS (Amazon Web Services) or Microsoft's Azure platform. This allows the system to more easily adapt to spikes in needed processing power. This also impacts the cost of the system since with the cloud services typically you only pay for what you use. Cost associativity is another important benefit of cloud services since paying for one server for 20 hours cost the same as paying for 20 servers for one hour. This aspect is especially important for batch related processing since if the results are needed more promptly than usual the elasticity of the cloud can be leveraged. Owning the hardware makes the adaptation to fluctuating loads more complicated since then an organization will have to over-provision its resource pool leading to idling resources. On the other hand while using cloud services one must take into account the cost and time of moving the data to and from the cloud.

4) Maintainability: By using a public cloud service, an organization does not have to make a big commitment for the hardware. However there are negatives such as business agility where a company might feel they are locked-in with the service. Conflicts might arise if the service provider changes its terms of service, increases the price or even shuts down completely. Deploying the computation engine and other services onto private cloud is an approach to relief the concerns about business agility, but increasing in the cost from maintaining hardware and re-deploying the computing engines is foreseeable.

5) Cost: Lowered costs are one of the main temptations associated with cloud migration. More specifically, the economic benefits come from paying for resources usage and operational costs separately, such as CPU time, memory usage, power, cooling, and other physical plant costs. There are many factors that need to be taken into account such as the system infrastructure cost, cost of the actual migration and costs associated with using the system. The skills of the developers also have to be taken into account since it might be a new platform that the developers have to learn which could cause longer time-to-market.

B. Challenges and lesson learned

During the progress of implementing the architecture, challenges had been encountered. In this section we share two main challenges, which are the segmentation of the system and the transformation of components.

1) Segmentation of the system: To separate the concerns, the monolithic system needs to be differentiated into smaller segments with clear boundaries. According to the functional coherent principle, we divided the components of the monolithic system into the logic layers in accordance to their responsibilities. The components interfacing with other systems were put into the integration layer, the components that do the computation work were put into the application layer and the components that handle the data storage were put into the data layer.

2) Transformation of components: To adapt the system into CloudSOA, components in the integration layer were made into service-consumer applications that listen to the data source and invoke the services accordingly. Data computing components in the application layer were transformed into reusable processing logic that can be used by the cloud computing engine. The data layer components were omitted since the data storage service will have to be accessed by its own interface. The cloud computing engine is a new element added to the system as a service, and the service bus acting as the communication backbone is a newly introduced element as well.

In general, service bus is the primary element in the architecture, where communication style could be chosen from messaging, publish-subscribe or others. Services in the architecture could be application platforms, such as the computing engine in this case, other general-purpose systems, like logging system, and other third-party services.

C. Threats to validity

The main threat to the validity of this report is the non-exhaustive evaluation due to two factors. First being that only a small part of the complex SPRINT system has been implemented, and the limited amount of hard data gathered. Therefore in-depth analysis and comparison of crucial factors such as cost and performance have been left out of this report. It is possible that there are requirements that have not been taken into consideration that renders CloudSOA not suitable. Finally, the basis of this report is to investigate how to cloudify a batch processing system, taking special regards to the systems scalability and viability. We have proposed one way of accomplishing this, however, it is important to stress that there are many alternative approaches.

VII. Conclusion

The primary objective of this study has been to investigate how to cloudify a legacy batch-processing system. Cloudifying is the process of moving a system to the cloud, this can bring benefits for an organization such as reduced cost, increased performance and faster results. The proposed architecture CloudSOA and its migration plan could be referred to in the case where an organization wishes to cloudify a legacy batch-processing systems and the cross-cutting concerns are appropriately addressed.

VIII. Future work

Future works need to be performed on CloudSOA, such as comparisons of performance running on the same hardware, cost comparisons. General integration guidance with upstream and downstream applications of the system could also be further studied. And generic service interface specification will also be valuable to interested audience.

IX. Acknowledgements

The authors would like to thank Mac Svan, Johan Axelsson, Per-Arne Lövgren and Michael Voemel from AB Volvo for their kind assistance. We would also like to thank Michal Palka and Imed Hammouda of Chalmers University for their invaluable guidance and feedback.

References

[1] M. van Sinderen, "Challenges and solutions in enterprise computing," *Enterprise Information Systems*, vol. 2, no. 4, pp. 341–346, 2008.

[2] K. Bennett, "Legacy systems: coping with success," *IEEE Software*, vol. 12, no. 1, pp. 19–23, 1995.

[3] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud migration research: A systematic review," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 142–157, 2013.

[4] R. Stephens and E. C. (e-book collection), *Beginning software engineering*, 1st ed. Indianapolis, Indiana: Wrox, 2015.

[5] J. Bingham, "Distributed systems," in *Mastering Data Processing*. Springer, 1983, pp. 235–245.

[6] K. Channabasavaiah, K. Holley, and E. Tuggle, "Migrating to a service-oriented architecture," *IBM DeveloperWorks*, vol. 16, 2003.

[7] S. Alahmari, E. Zaluska, and D. De Roure, "A service identification framework for legacy system migration into soa," 2010, pp. 614–617.

[8] H. M. Sneed, "Integrating legacy software into a service oriented architecture." IEEE, 2006, pp. 11 pp.–14.

[9] ——, "Migrating from cobol to java," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–7.

[10] ——, "A pilot project for migrating cobol code to web services," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 11, no. 6, pp. 441–451, 2009.
[11] M. Razavian and P. Lago, "A systematic literature review on soa migration," *Journal of Software: Evolution and Process*, vol. 27, no. 5, pp. 337–372, 2015.
[12] M. Colosimo, A. D. Lucia, G. Scanniello, and G. Tortora, "Evaluating legacy system migration technologies through empirical studies," *Information and Software Technology*, vol. 51, no. 2, pp. 433–447, 2009.
[13] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," pp. 50–58, 2010.
[14] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004.
[15] I. IEC, "25010 (2011) systems and software engineering-systems and software quality requirements and evaluation (square)-system and software quality models," *International Organization for Standardization, Geneva, Switzerland*.
[16] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton, "Reference model for service oriented architecture 1.0," *OASIS standard*, vol. 12, p. 18, 2006.
[17] 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*. USENIX Association, 2012, pp. 2–2.
[18] T. D. A. D. J. M. M. M. M. J. F. S. S. I. S. U. o. C. B. Matei Zaharia, Mosharaf Chowdhury, "A fault-tolerant abstraction for inmemory cluster computing," *Datasets, Resilient Distributed*.
[19] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.