Towards Hyperscale Process Management

Kevin Andrews[1], Sebastian Steinau[1] and Manfred Reichert[1]

Abstract: Scalability of software systems has been a research topic for many years and is as relevant as ever with the dramatic increases in digitization of business operations and data. This relevance also applies to process management systems, most of which are currently incapable of scaling horizontally, i.e., over multiple servers. This paper discusses an approach towards hyperscale workflows, using a data-centric process engine to encapsulate data and process logic into objects, which can then be stored and concurrently manipulated independently from each other. As this allows for more concurrent operations, even within a single data-intensive process instance, we want to prove that an implementation of a hyperscale process engine is a feasible endeavor.

Keywords: Scalability, Process Management Technology, Object-centric Processes

1 Introduction

For decades, researchers have been examining parallelism and scalability in computer hard- and software. The topic of scalability also became instantly relevant to workflow management systems (WfMS) when they first showed up on the market, as they were built explicitly with large-scale applications in mind. First attempts to create scalable workflow management systems applied existing scalable architecture principles to WfMS. The resulting approaches, such as WIDE [CGS97], OSIRIS [Sc04], and Gridflow [Ca03], strongly focused on the system architecture point of view, largely ignoring other aspects, such as role assignments, permissions, and data flow. However, the process models these approaches, especially Gridflow, are meant to support, are typically high-performance compute workflows, where these aspects merely play a secondary role.

Due to these limitations, as well as further advances in processing power over the years, most modern WfMS and process engines for human-centric "business" processes do not offer horizontal scalability. This is likely acceptable with modern computing power in most company intranet scenarios. According to Amdahl's law of scalability [Am67], which assumes that more processing power and more processors equal lower turnaround times for a fixed workload, this would mean that there are no more, or only negligible, scalability problems in WfMS. However, Gustafson's law [Gu88], a reevaluation of Amdahl's law, states that the workload scales upwards with the increases in resources and performance, meaning that the time needed to execute a workload stays virtually the same with more computational power. This effect can be seen in the trend towards cloud-based software, including cloud-based process engines. The workload placed on a cloud-based process engine is higher than the one placed on an on-premise WfMS ten years ago, just as the hardware capabilities of a server hosting such a cloud-based process engine are higher.

[1] Institute of Databases and Information Systems, Ulm University, firstname.lastname@uni-ulm.de

In summary, this means that improving process engine scalability can offer benefits to performance and cost reduction in data-centers. These advancements can help pave the way for better market penetration of cloud-based process engines and thereby also process management technology as a whole, even in smaller companies.

To facilitate this, we present the idea of replacing the typical activity-centric conceptual basis with a data-centric one. We intend to show that, besides their many other benefits [KWR11], data-centric approaches are better suited for highly concurrent work with processes that are data-heavy and involve large amounts of user interaction. Section 2 explains some fundamentals of the data-centric process management approach that we intend to use as the basis for our research. In Section 3, we compare the scalability possibilities of activity-centric process models to our vision of parallelization with data-centric models. Finally, Section 4 gives an outlook, based on the findings presented in this paper.

2 Fundamentals

This section gives a short overview of PHILharmonicFlows [KR11, KWR11], an object-aware approach to business process management. The PHILharmonicFlows approach was selected as a basis for our research on scalability as its core idea is to group data and associated processes into *objects*. These objects are largely independent units and can be interacted with individually. One such object exists for each business object present in a real-world business process. As can be seen in Fig. 1, a PHILharmonicFlows object consists of data, in the form of *attributes*, and a process model describing the object *lifecycle*.

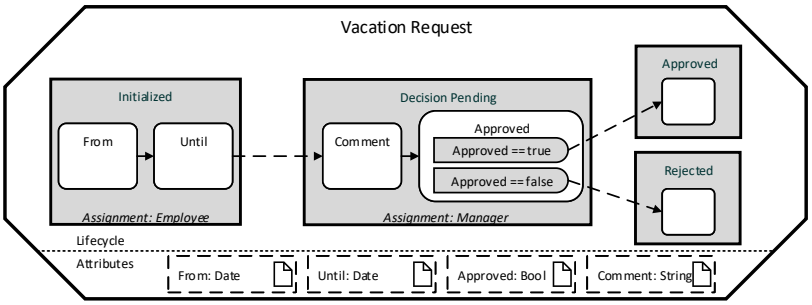

Fig. 1: Example PHILharmonicFlows Object

The attributes encapsulated in the *Vacation Request* object are *From*, *Until*, *Approval*, and *Comment*. The *lifecycle process* describes the different *states (Initialized*, *Decision Pending*, *Approved*, and *Rejected)*, a *Vacation Request* may have during process execution. As PHILharmonicFlows is data-driven, the lifecycle process for the *Vacation Request* can be understood as follows: The initial state of a *Vacation Request* object is *Initialized*. Once an *Employee* has entered data for the *From* and *Until* attributes, the state changes to *Decision Pending*, which allows a *Manager* to input data for *Comment* and *Approved*. Based on the value for *Approved*, the state of the *Vacation Request* changes to *Approved* or *Rejected*.

Based on the current state of an object, it can be coordinated with other objects corresponding to the same business process through a set of constraints, defined in a separate *coordination process*, details of which are omitted for brevity. A simple example could be a constraint stating that a *Vacation Request* may only change its state to *Approved* if there are less than 4 other *Vacation Requests* already in the *Approved* state. A complete process consists of many different objects, each describing data and the different states the object may enter. These excerpts from PHILharmonicFlows will be used in the remainder of the paper to show that a data-centric process engine can be used for highly scalable processes.

3 Methodology

Our research idea stems from the examination of past attempts at creating highly scalable WfMS. As mentioned in Section 1, most of these approaches [CGS97, Ca03, Sc04] focus on industrial or scientific workflows and not on processes with a large amount of user interaction. The approach presented in [BRD03] proposes partitioning a process model based on role assignments and executing these partitions on different servers so that the network and processing load of a single process instance can be distributed. However, this approach has significant drawbacks in respects to its flexibility, as the partitioning is done at build-time and is limited by the structure of the model. Consider the abstract activity-centric process model shown in Fig. 2.

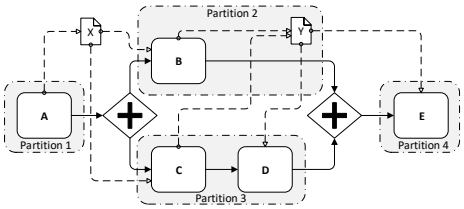

Fig. 2: Partitioned Activity-centric Process Model

If the process model is partitioned in this way, i.e. according to the role assignments of the different activities, Partitions *2* and *3* can be executed in parallel, increasing scalability. However, a maximum of two tasks can be executed at the same time, limited by the number of branches in the process model. Furthermore, the depicted data-flow introduces copy-operations of data between nodes on control flow splits and joins, as well as other problems, such as lost updates. In summary, the process modeler has additional non-trivial concerns when creating the process model.

We propose to alleviate these issues by relying on a data-centric approach for executing the workflows. To illustrate the idea, we use the PHILharmonicFlows concept (cf. Section 2). As shown in Fig. 1, PHILharmonicFlows allows grouping data and lifecycle information into objects. Expanding the previous example, this means that it is possible to have *n Vacation Request*, and also other objects, at run-time, each having their own attribute values and lifecycle processes. In consequence, as these lifecycle processes need only be coordinated at certain state changes, they can be executed in parallel.

Fig. 3 shows an illustration of how n clients can interact with the m objects existing at run-time. The objects are largely independent of one another, except for when their states change. Furthermore, the n objects and, therefore, the data and computational resources needed to serve the m users, can be partitioned over n systems.

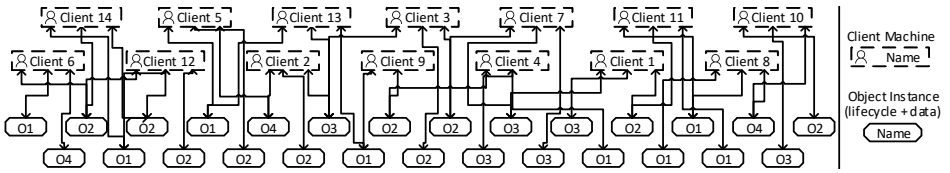

Fig. 3: Example PHILharmonicFlows Process Instance with Clients

We compare these two basic ideas, involving either an activity- or a data-centric theoretical basis, by applying Gustafson's law to them. Gustafson's law (1) states that the theoretical speedup *S(n)* for n processors can be calculated using the fraction of the problem that can be executed in parallel, p, as well as the fraction that must be executed in serial, *(1-p)*.

$$S(n) = (1 - p) + n \times p \tag{1}$$

While calculating factor p for a given activity-centric process model is theoretically possible through analyzing the model and determining the fraction of parallelizable tasks, n will always be limited to the average amount of branches that can be executed in parallel. The exact values for p and n are not relevant, which may be illustrated for a trivial case by examining the process model from Fig. 2. The value for p is around $\frac{3}{5}$, depending on the applied metrics, while the maximum value for n is between 1 and 2, also depending on the exact metrics. So, assuming we have 1 processor executing the process model, this means that we obtain our sequential base speed $S(1) = (1 - \frac{3}{5}) + 1 \times \frac{3}{5} = 1$, i.e. 100%. For an approximate maximum n of $\frac{3}{2}$, the speedup factor $S(\frac{3}{2}) = (1 - \frac{3}{5}) + \frac{3}{2} \times \frac{3}{5} = 1.3$, i.e. 130% of the base speed. As n, for the process model depicted in Fig. 2, can never reach the value 2 or above, and the speedup factor for $n = 2$ is 160%, it is obvious that activity-centric process models are limited in their scalability, which is, in turn, determined by the structure of the process model itself.

These calculations, however, are very different for approaches like PHILharmonicFlows, where n is not limited by the structure of the individual objects. As the entire business process consists of an unlimited number of objects, the limit for n is the amount of object instances present in a process instance. Clearly, if only 2 objects are in use at run-time, the maximum value for n is 2, however, the maximum value for n scales upward with each object present in a process instance. There is, however, a limitation to the p factor, as the coordination of the objects' interactions among each other is handled by one or more *coordination processes*, which are notified when an object changes its state. The exact value of p depends on the amount of rules defined in the coordination process, but even assuming that only 10% of the work can be executed in parallel and that there are 100 objects present in a process instance, the speedup $S(100)$ is $(1 - \frac{1}{10}) + 100 \times \frac{1}{10} = 10.9$, i.e. 1090%. For a more realistic value of $p = 0.8$ the expected speedup increases to 8020%.

Obviously, even for very inefficiently structured PHILharmonicFlows processes, there is a far greater potential speedup from scaling over multiple machines and processing cores than with an activity-centric approach. An important fact to note is that the scaling possibilities do not only increase with the hardware capabilities, but also with increased load on a single process instance, i.e., more clients and more objects created. Naturally, these are rough calculations based on simple process models, which will have to be examined in greater detail, both theoretically, as well as empirically in different practical scenarios.

4 Outlook

Based on the idea presented in this paper, we plan on further examining the viability of data-centric approaches for creating a hyperscale process engine. Furthermore, we plan on taking the existing PHILharmonicFlows process engine implementation and calculating the optimal partitioning of its conceptual elements into individual micro-services. Ideally, we will be able to determine a partitioning schema ensuring the best values for all relevant performance indicators, such as increased network and communication and load balancing efficiency. Finally, once the theoretical groundwork is established, we hope to create a working prototype of hyperscale process engine based on PHILharmonicFlows. The engine we envision should be capable of running in the cloud and servicing a large number of concurrent interactions with a single process model.

References

[Am67] Amdahl, Gene: Validity of the single processor approach to achieving large scale computing capabilities. In: AFIPS Spring Joint Comp Conf. pp. 483–485, 1967.

[BRD03] Bauer, Thomas; Reichert, Manfred; Dadam, Peter: Intra-subnet load balancing in distributed workflow management systems. Intl J of Coop Inf Sys, 12(03):295–323, 2003.

[Ca03] Cao, Junwei; Jarvis, Stephen A; Saini, Subhash; Nudd, Graham R: Gridflow: Workflow management for grid computing. In: 3rd IEEE/ACM International Symp on Cluster Comp and the Grid. pp. 198–205, 2003.

[CGS97] Ceri, Stefano; Grefen, Paul; Sanchez, Gabriel: WIDE - A distributed architecture for workflow management. In: 7th Intl Workshop on Research Issues in Data Engineering. IEEE, pp. 76–79, 1997.

[Gu88] Gustafson, John: Reevaluating Amdahl's law. Comm of the ACM, 31(5):532–533, 1988.

[KR11] Künzle, Vera; Reichert, Manfred: PHILharmonicFlows: towards a framework for object-aware process management. J of Soft Maintenance and Evolution: Research and Practice, 23(4):205–244, 2011.

[KWR11] Künzle, Vera; Weber, Barbara; Reichert, Manfred: Object-aware business processes: Fundamental requirements and their support in existing approaches. Intl J of Inf Sys Modeling and Design, 2(2):19–46, April 2011.

[Sc04] Schuler, Christoph; Weber, Roger; Schuldt, Heiko; Schek, H-J: Scalable peer-to-peer process management - the OSIRIS approach. In: IEEE Intl Conf on Web Services. pp. 26–34, 2004.