Datenbank Spektrum (2015) 15:99–108
DOI 10.1007/s13222-015-0186-1

SCHWERPUNKTBEITRAG

Optimized P2P Data Management for Reliable Workflow Execution in Mobile Environments

Nenad Stojnic · Heiko Schuldt

Received: 3 March 2015 / Accepted: 28 April 2015 / Published online: 16 June 2015

Abstract Workflows allow to build complex applications out of existing services. Hence, workflows are inherently distributed. At the same time, workflows are mostly executed by centralized workflow engines that invoke the services of a workflow in a request/reply style. These engines may become a performance bottleneck and limit the scalability of the entire system. As a consequence, approaches to distributed workflow execution have been proposed. While these approaches have better scalability characteristics, they require additional efforts to deal with (partial) failures of the system. To cope with such failures, workflow instance data must be stored redundantly at different sites. This is even more important in mobile environments or mixed mobile/stationary environments where the nodes hosting services for workflows may be mobile and are thus more likely to fail permanently or become temporarily unavailable. This is the case, for instance, for sensor net applications where mobile devices capture data, or when smartphone apps share and jointly process data.

In this paper, we present the combination of OSIRIS-SR, an extension to the distributed and decentralized workflow engine OSIRIS that focuses on reliable workflow execution by means of instance data replication, and Compass, an extension to Chord that is particularly tailored for efficient P2P data management on mobile devices.

Keywords Distributed workflow management · P2P data management · Mobility

N. Stojnic · H. Schuldt (✉)
Databases and Information Systems Group, University of Basel, Basel, Switzerland
e-mail: heiko.schuldt@unibas.ch

1 Introduction

In the last decade, service-oriented architectures have been established as an important pattern for building large-scale applications. Complex services, also called workflows, can be recursively defined by means of other services. Moreover, these services can be distributed across several nodes in a network. Consequently, workflows are inherently distributed applications. Most current approaches to workflow execution rely on a centralized workflow engine that invokes the services of a workflow in a request/reply style. However, due to their centralized architecture, these engines may become a performance bottleneck and limit the scalability of the entire system.

Recently, approaches to distributed workflow execution have been proposed where workflow engine functionality is distributed across several nodes in a network. While these approaches provide better scalability characteristics, they require additional efforts to deal with (partial) failures of the system, especially failures of individual nodes on which workflow execution currently takes place. To cope with such failures, workflow instance data must be stored redundantly at different sites.

When mobile devices or mixes of mobile and stationary devices are used for workflow execution (for instance in smartphone apps where data is shared and collectively processed by mobile devices, or in sensor networks where mobile devices are used for capturing data—such as in emergency assistance [5] or health monitoring [7]), dealing with permanent or temporary failures of nodes hosting services for workflows is even more important. Failure of a device currently in charge of executing a workflow might result in the loss of complete workflow instances. Hence, the execution state of workflows needs to be replicated across different sites (mobile and/or stationary).

In our previous work, we have introduced a safety ring for the reliable, distributed and decentralized execution of workflow instances. The safety ring has been integrated into the OSIRIS (Open Service Infrastructure for Reliable and Integrated process Support) distributed workflow engine [18, 19]. In this paper, we show how OSIRIS' safety ring, called OSIRIS-SR [22], can be augmented with support for efficient data replication in an enhanced peer-to-peer (P2P) overlay, called Compass [21], that is explicitly tailored to data management on mobile devices. Compass extends existing distributed hash tables (DHTs) like Chord which provide efficient access to data, but which are completely agnostic to different node capabilities, including the nodes' bandwidth and latency characteristics. These DHTs assume that all the nodes and their network connections in a P2P system are homogeneous in nature. If this is not the case, especially in heterogeneous, mobile P2P systems that consist of both mobile and fixed nodes with significantly differing bandwidth and latency characteristics, optimizing the number of hops in a DHT for data access is no longer a feasible solution. Compass addresses such environments and aims at optimizing data access by reducing the overall latency. This approach also inherently supports the mobility of nodes which results in dynamically changing connection characteristics. The combination of Compass and OSIRIS-SR therefore provides reliable distributed workflow execution by replicating instance data in a P2P system and allows to optimize access to these instance data by taking into account the network latency of the nodes where replicas are stored. Moreover, we present the evaluation of OSIRIS-SR that shows the effective failure handling and the gain in efficiency of the Compass extension when dealing with mobile nodes.

The paper is organized as follows. Sect. 2 motivates the approach with a sample use case. OSIRIS and its safety ring extension OSIRIS-SR are introduced in Sect. 3. The Compass P2P overlay is presented in Sect. 4 and the integration of OSIRIS-SR and Compass is detailed in Sect. 5. Sect. 6 shows evaluation results of OSIRIS-SR and COMPASS. In Sect. 7, we survey related work and Sect. 8 concludes.

2 A Sample Use Case

Consider, for instance, an emergency management scenario in which a group of firefighters is sent out into the field to jointly extinguish a fire. Triggered by various environmental changes, critical situations (e.g., sudden change of wind direction) can occur that put the firefighters into risk. Since their individual perception may be limited due to their hostile environment, they most probably cannot anticipate such critical situations. In order to reduce the risk in a traditional sense, firefighters rely on the coordination with their headquarters by means of rather bulky communication devices.

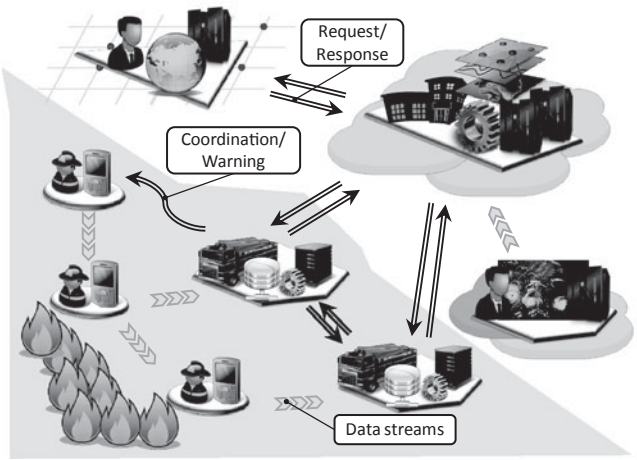

Fig. 1 Sample application: Modern emergency management

However, given the high dynamics of their tasks and the communication environment they use, it is not always possible for them to directly communicate with each other and thus to share information on their environment in a timely manner.

As a vision for the future, consider a wearable intervention support system. This system exploits a variety of sensors that are spread out in the action area or are attached to the firefighters' equipment. Among others, sensors capture the physical state of the firefighters, their actions, their location, the environmental state of the action area, etc. In conjunction with other sources (e.g., weather forecasts), the data produced by the various sensors is subject to sophisticated analyzes with the goal of anticipating critical situations.

Such analysis is usually done by dedicated workflows on the basis of rather simple services. The outcomes of the data analysis workflows, which should be executed in near real-time and thus close to the area of action, are for instance danger area heat-maps. Therefore, all emergency staff should not only be equipped with sensors (such as cameras, GPS sensors, physiological sensors) but also with smartphones or other mobile data processing devices. These devices should run simple services (e.g., sensor data filtering) and should have a wireless connection with more powerful stationary devices, for instance in the nearest fire truck, where more resource-demanding services can be hosted. Complex analysis processes should be reliably executed on these mobile and stationary devices, without dependency on any central device that may become a single point of failure or a potential performance bottleneck. It is important to process data close to where they are created and to apply failure handling to make sure that vital analysis is not impacted by the failure of individual devices. Results of these analysis workflows have to be disseminated immediately among all emergency staff and also be transmitted to their headquarters. Fig. 1 provides an illustration of this emergency management scenario.

3 System Model

In this section, we briefly introduce the architecture of the distributed OSIRIS workflow engine and the OSIRIS-SR extension for reliable workflow execution.

3.1 OSIRIS

A major feature of OSIRIS is its inherent distribution: in contrast to most existing workflow execution engines which orchestrate distributed services from a centralized engine, also the OSIRIS engine itself and thus workflow execution is distributed across a set of cooperating nodes. Based on a distinction between *application services* and *system services*, workflow execution is provided by a set of system services acting as 'mini workflow engines', so-called OSIRIS layers, available on each node in the system. Application services, in turn, provide the functionality that is invoked by workflows. The OSIRIS layers are able to locally invoke an application service in the context of a workflow execution and then forward control to the OSIRIS layer of another node where the next service according to the workflow definition is hosted. This workflow execution requires that each node equipped with an OSIRIS layer has the necessary metadata to locally make a routing decision and to select the next service to forward control to. In case the service to be invoked is only provided externally, i.e., on nodes that are not hosting the OSIRIS layer, a proxy service on one of the OSIRIS layers is used to invoke the external services [6].

In order to make this metadata locally available, OSIRIS exploits a dissemination model based on the publish/subscribe paradigm. Each provider of an application service subscribes for the necessary metadata its OSIRIS layer needs for executing the part of the workflow in which the local application services are involved. This includes a list of services (per service and per workflow in which this service is to be invoked) to be executed after the local invocation, a list of providers of these services, and also the load of these providers to apply load balancing. Due to the publication of relevant metadata according to a node's subscription, this information is maintained locally by each OSIRIS layer and does not have to be queried at run-time. Hence, workflow execution can proceed in a P2P style, without any centralized component.

For this, additional system services need to be provided. These include global system repositories that maintain metadata on the overall system such as workflow definitions, service providers, their load, and subscription lists. These system services are complemented with publish/subscribe functionality to make sure that all relevant nodes (according to their subscription) are immediately informed about relevant changes of the metadata they need. The run-time routing decisions are completely decoupled from metadata replication as this process happens even before the workflow execution starts. Late binding of services, i.e., the selection of service providers at run time, in conjunction with load balancing strategies, distributes the overall workload across the available service providers and thus avoids potential bottlenecks or single points of failure and allows for a high degree of scalability.

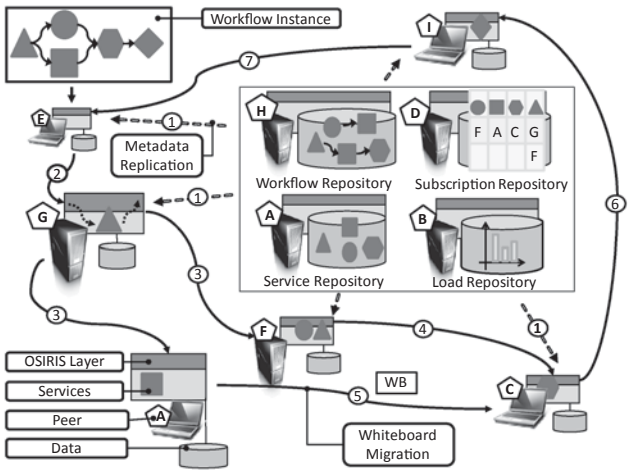

Fig. 2 Distributed workflow execution with OSIRIS

When the OSIRIS layer of a node forwards control after a local invocation of an application service to the OSIRIS layer of another node for the purpose of continuing workflow execution, the state of this workflow (instance data) is serialized and stored in a so-called whiteboard, with is transferred to the new node together with the workflow execution control. This also ensures proper data flow between successive service invocations within a workflow.

To define workflows, OSIRIS relies on standard technologies such as WS-BPEL [3]. Moreover, OSIRIS supports two alternative approaches to workflow specification: first, O'GRAPE (OSIRIS GRAphical Process Editor), a graphical workflow definition tool [23] that also allows to specify the transactional semantics of workflows [17], and second, OSIRIS-PRO, a programming-in-the-large approach to workflow definition that uses a JavaScript like workflow language.

The architecture of OSIRIS and the process of executing a workflow is depicted in Fig. 2. The numbers indicate the sequence of metadata dissemination and forwarding of control flow, while the letters distinguish the providers of both system and application services (the latter are illustrated with different shapes). Further details on the publish/subscribe replication mechanism in OSIRIS can be found in [18, 19].

3.2 OSIRIS' Safety Ring

While the distributed architecture of OSIRIS provides a high degree of scalability, it is at the same time also vulnerable to

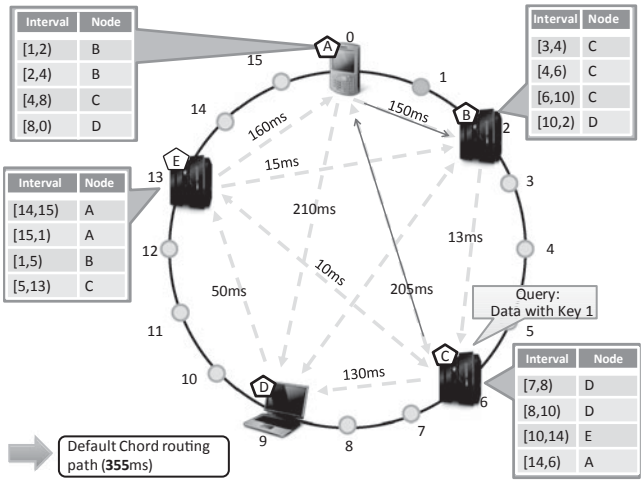

Fig. 3 Standard Chord routing

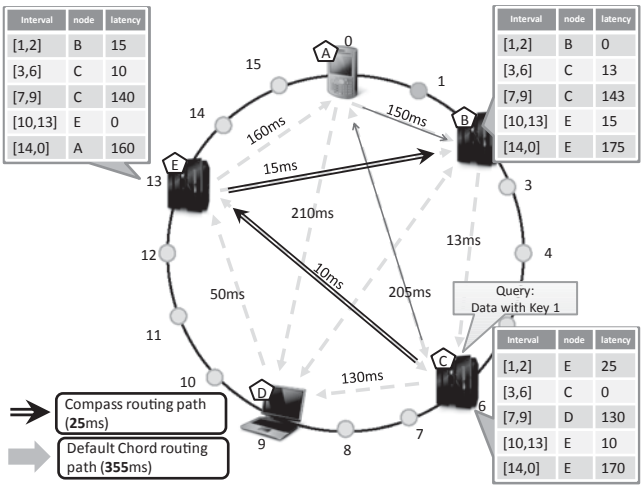

Fig. 4 Optimized Compass routing

the failure of single nodes. Especially the failure of a node which is currently in charge of executing parts of a workflow might lead to the complete loss of a workflow instance, as the control information as well as the whiteboard will disappear. Hence, dedicated monitoring and replication mechanisms are needed to detect the failure of nodes and to apply failure handling to make sure that no single piece of information gets lost.

In order to overcome these drawbacks, a further system service, called *safety ring* has been introduced [22]. OSIRIS-SR, the safety ring extension of OSIRIS, provides two additional features: first, it monitors other nodes to detect failures (by periodically probing the monitored service providers using heart beat messages) and to initiate failure handling. Second, it replicates whiteboard data in the system to ensure correct recovery. Replication of the whiteboard takes place across a small set of safety ring nodes, called the replica pool. Out of this pool of redundant safety ring nodes, one is selected as leader. This leader l actively monitors a regular OSIRIS node o and takes over control in case node o fails. The other safety ring nodes act as backup for l in case it fails as well. The whiteboard data is replicated in a P2P style, by means of a Chord-based DHT overlay structure [20] which is added on top of all the nodes in the system.

The failure of o is detected by l with the help of an eventually perfect fault detector [8] and handled by means of re-invocation. For this, the leading safety ring node l provides another node that has not failed and that hosts the same service type with a back-up copy of the whiteboard and thus seamlessly resumes workflow execution there. In the event of a failure of l, the Chord DHT repairs itself by re-assigning the leader role to the succeeding safety ring node l' in the topology. The new leader l' restores the whiteboard replication factor by copying the whiteboard to other members of the replica pool. Moreover, l' resumes monitoring of the nodes that have been invoked with the copy of the whiteboard. The safety ring mechanism works only as long as there are enough safety ring nodes and service providers available to substitute the failed nodes. Further details on the safety ring mechanics can be found in [22].

4 Optimized Peer-to-Peer Data Management

Data management in P2P systems is usually based on overlay networks that exploit a DHT. The most prominently used DHT implementation is Chord [20]. While DHTs like Chord optimize data access by minimizing the hops needed to reach the node in the P2P system hosting a particular data item, these DHTs are at the same time agnostic to different node capabilities, in particular to different bandwidth and latency characteristics. In a homogeneous environment, optimizing the number of hops for data access is a feasible solution. But this is no longer the case in heterogeneous, mobile P2P systems that consist of both mobile and fixed nodes where the bandwidth and latency characteristics of nodes significantly differ. Moreover, due to the mobility of nodes, connection parameters can dynamically change. In such environments, routing should aim at optimizing data access by reducing the overall latency.

Compass [21] has been introduced as an extension to Chord that is particularly tailored to mobile environments. In addition to Chord's finger table, each peer hosts an additional data structure called Compass table. These tables are built and continuously updated by probing the network and thus dynamically adapt to changing node characteristics.

As an example, consider the Chord ring depicted in Fig. 3. A query for data identified by key 1 issued at peer C will be routed, according to the Chord protocol and the information in the local finger tables, via node A to node B where the

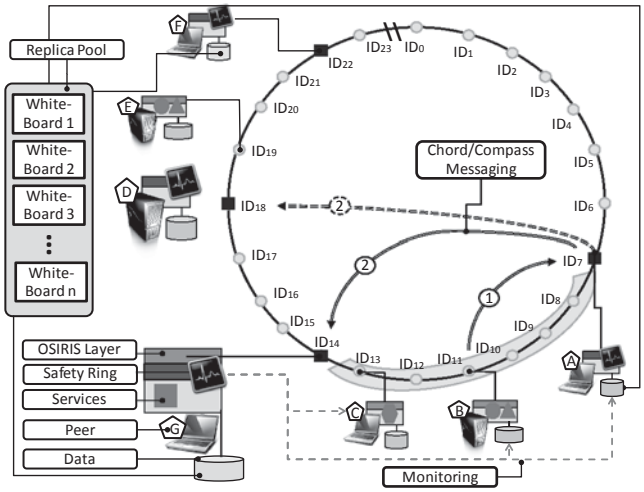

Fig. 5 The OSIRIS safety ring for mobile environments

data can be accessed. However, node A happens to be a mobile node. Even though the number of hops is optimized, the latency for accessing data on node B via A is quite high, due to the connection characteristics of the intermediate mobile node A.

Compass follows a latency-aware approach to optimize data access on top of Chord. It dynamically determines the data access paths that minimize overall latency—which are are predominantly composed of powerful (i.e., static) nodes—while preserving the fundamental routing features of Chord such as scalability and consistency. For this, the Compass tables contain routing information which is derived from the access latencies to the finger table nodes. The Compass table initialization starts with a latency value for each identifier in the network. This table is then incrementally compressed by identifying adjacent key identifier intervals with the same latency values which are then combined in an enlarged identifier interval. The result is a table which maintains lists of triples that comprise the interval in the identifier space, the node to be contacted for this interval, and the latency value for data access. Note, that similar to the Cord finger tables, the destination node of a Compass table might be an intermediate node that further forwards the request (again by optimizing the access latency).

The Compass table is dynamically maintained to take into account node dynamics (new nodes joining, existing nodes leaving). Moreover, the intervals and the nodes responsible for intervals are continuously adapted when access latencies of a node in the system change (mainly because of mobility). Again, this is done by continuously probing nodes in the network. More details on the initialization and maintenance of Compass tables are given in [21].

Fig. 4 shows the Compass extension to the example of Fig. 3. While the ring topology is maintained, the Compass tables may consider different intervals than the Chord finger

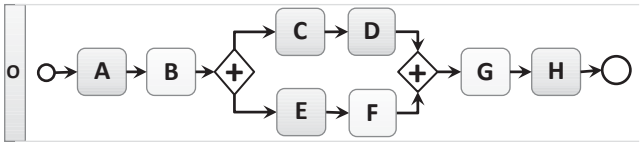

Fig. 6 Sample evaluation workflow. Lighter boxes denote computationally intensive activities of the workflow

tables. As a result, the routing paths using the Compass tables might differ from those given by the Chord finger tables. In the example query for data with identifier 1 issued at node C, the latency-optimized path also includes two hops, but with a significantly lower overall latency (25 ms with Compass compared to 355 ms with standard Chord) as the request will be routed via stationary node E.

5 OSIRIS-SR for Mobile Environments

OSIRIS-SR for mobile environments combines the safety ring for node monitoring and whiteboard replication with a DHT overlay for the nodes in the system that use the Chord protocol with Compass extension. For this, the safety ring system service needs to be deployed on several hosts; it might even be available on all nodes of the system, together with the OSIRIS layer, to balance the monitoring and whiteboard data replication effort in the system. All safety ring nodes are mapped to a logical ring overlay. Fig. 5 depicts the OSIRIS-SR node topology. Node G with ID_{14}, for instance, on which the safety ring service is installed, acts as a monitor for nodes in the area marked in grey (in the example of Fig. 5, these are service providers C and B). Assume that node B which provides application services and which takes care of workflow execution by its local OSIRIS layer needs to be monitored. For this, r nodes in the virtual overlay that locally provide a safety ring system service will be selected. In the example in Fig. 5, the replication factor is $r = 3$ out of in total four safety ring nodes (A, D, F, and G) and the monitoring nodes are nodes D, F, and G. Having more than one safety ring node for monitoring makes sure that failures of node B can be handled even in case of the simultaneous failure of a safety ring node. Whiteboard replication in the safety ring uses symmetric P2P replication [9]. If node G, the active monitoring node, fails, then its role will be taken over by the next safety ring node in the identifier space (node D), and a new safety ring node will be selected to keep the replication factor.

These safety ring nodes not only monitor 'normal' nodes, they also keep replicas of the whiteboard that contains the execution metadata of the workflow instances of these nodes. Changes to the whiteboard will be stored at the safety ring nodes that monitor the node that is currently in charge of workflow execution. Access to the whiteboard is provided

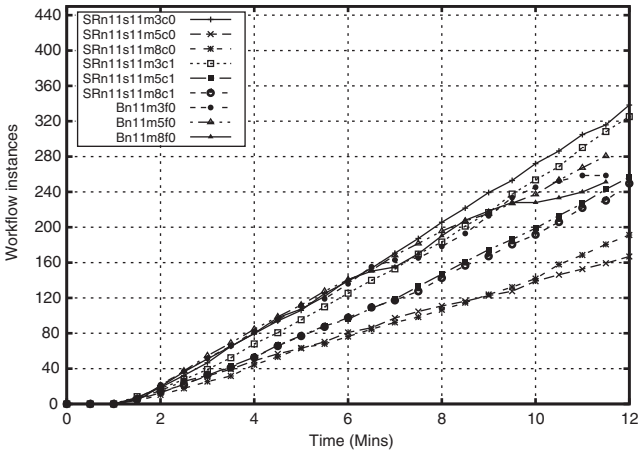

Fig. 7 OSIRIS-SR evaluation, 11 nodes, w/o failures

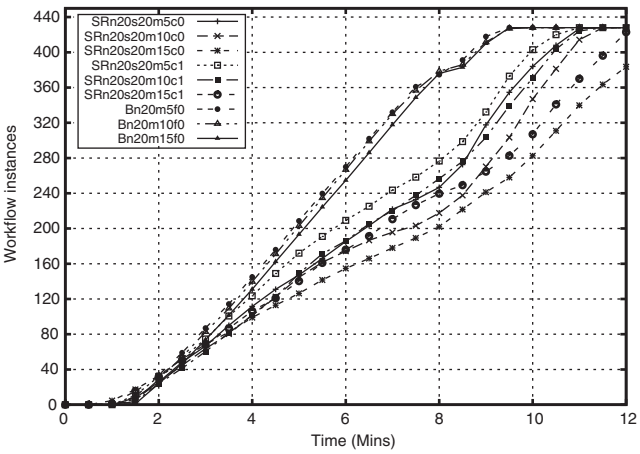

Fig. 8 OSIRIS-SR evaluation, 20 nodes,w/o failures

by the Compass protocol. In the example depicted in Fig. 5, node *B* accesses the whiteboard (after consulting one of the safety ring nodes, in our example node *A*) via the Compass protocol that minimizes latency (steps 1 and 2).

Whenever the local invocation of an application service has terminated, control is forwarded to the OSIRIS layer of the subsequent node according to the workflow description. In addition, the whiteboard has to be transferred. At the same time, the whiteboard will be replicated at the safety ring nodes. In order to guarantee that the transferred whiteboard together with the replicas stored in the safety ring are consistent, the Paxos protocol [10] is used for atomic commitment.

6 Evaluation

The evaluation of OSIRIS-SR has been performed using Amazon EC2 nodes of type c1.medium[1]. In order to simulate heterogeneous node environments, we transform EC2 nodes into failing, fluctuating, low bandwidth mobile nodes that are classified into three categories:

- *Static*. Static nodes are the plain EC2 virtual instances without any alterations.
- *Mobile*. A mobile node is simulated by artificially decreasing the network transmission performance of a static node. This is achieved by attaching dummy data of variable size to each message that is sent out into the network. The actual amount of data to be attached is linear with the physical distance towards the destination nodes. Node proximity is subject to dynamic change, thus the dummy data sizes dynamically vary within the bounds of 0 –100 kB.
- *Faulty*. A faulty node is simulated by stopping the OSIRIS processes at the corresponding static node at a random point in time. The precise stopping moment is determined by a uniformly distributed random function that remains within the interval 0–320 s. Each of the stopped nodes eventually rejoins the execution environment after 60 s of downtime and may henceforth again be subject to a random stop.

The workflow that is used for the evaluation is depicted in BPMN[2] notation in Fig. 6. The depicted workflow encompasses two types of activities: computationally 'moderate' and computationally 'intensive' ones. The computationally intensive activities are depicted with the lighter boxes and consume significantly more CPU resources at the nodes in conducting more complex and longer lasting numerical computations as compared to the other type.

To achieve a true P2P execution, only one activity type is deployed per node (so that the subsequent service of a workflow is hosted at another provider) and no node is capable of locally hosting more than one application service type. We have run two series of evaluations, the first with in total 11 nodes, the second with 20 nodes. The number of safety ring nodes has been varied in the evaluation, from 30 % to 100 % (in the latter case, this means that the safety ring system service is deployed on each node of the system).

The evaluation is conducted based on a user request queuing model that instantiates every Δt milliseconds new workflows and measures the throughput of the entire system. To elicit the scalability characteristics of the system, the number of started workflows is increased by decreasing Δt every minute by 15 %. We set the initial t value to 1.2 s which corresponds to 50 instances in the first minute. The load on the system will be steadily increased for the subsequent four minutes upon which in the final sixth minute we will maintain the same load as in the fifth minute and thus create a load plateau. Around 430 workflows will be launched against the

[1] http://aws.amazon.com/ec2/instance-types/.

[2] Business Process Model and Notation, www.bpmn.org/.

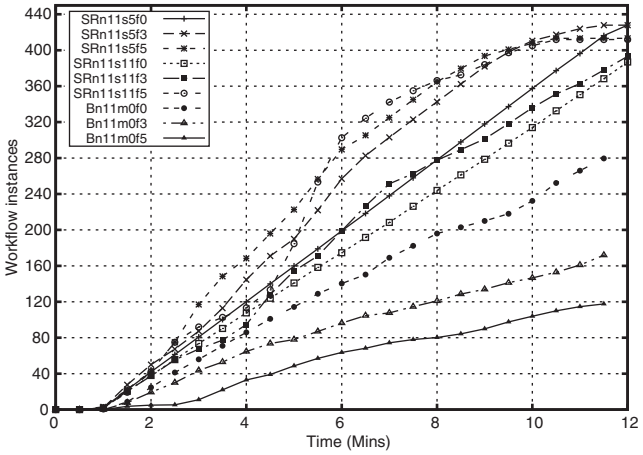

Fig. 9 OSIRIS-SR evaluation, 11 nodes, with failures

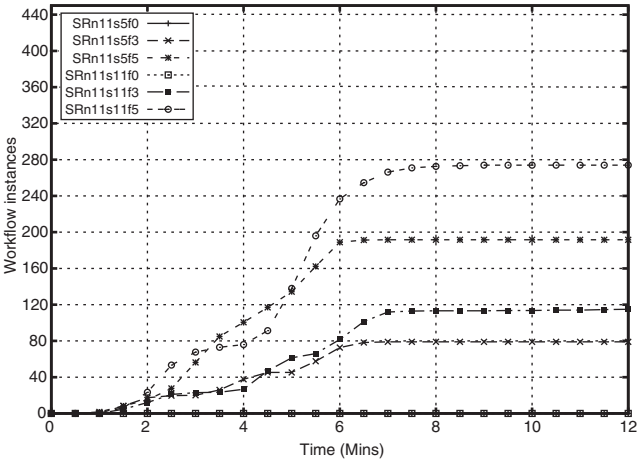

Fig. 10 OSIRIS-SR evaluation, 11 nodes, with failures and aborted workflows

system in total. To accommodate for failure and overload of nodes, we wait an additional four minutes to give lagging workflows a chance to catch-up.

In all evaluations, we have compared a baseline set-up (distributed workflow execution with OSIRIS, but without the safety ring's support for reliable execution) with OSIRIS-SR.

Fig. 7 shows the evaluation of OSIRIS-SR with Compass in the 11 node setting. The first six measurements (prefix SR in the chart's legend) correspond to the safety ring extension; out of these, the first three do not use the Compass extension but standard Chord, while graphs 4–6 use the Compass optimization for routing. The last three graphs (prefix B) correspond to the baseline set-up. All evaluations are done in a failure-free environment. In the evaluation runs, either three (m3), five (m5) or eight (m8) mobile nodes have been considered. Fig. 8 shows a similar evaluation in the 20 node setting (with 5, 10, and 15 mobile nodes).

Figures 7 and 8 allow to compare the evaluation results of the Compass protocol with the results obtained by standard Chord. The Compass version provides a throughput that is up to 30 % higher than that of standard Chord in the 11 node setting when a high degree of mobility is present. In the 20 node setting, the benefit is not as high as 30 % but still at around 10 %. Note that the differences between Compass and Chord are rather negligible when only a small set of nodes is mobile.

The effects of failures in the 11 node set-up are depicted in Fig. 9. Again, the six graphs in Fig. 9 with prefix SR correspond to OSIRIS-SR while the last three with prefix B correspond to the OSIRIS baseline set-up. In the first three graphs, the safety ring system service is deployed on 5 out of 11 nodes while in graphs 4–6, the safety ring system service is deployed on all nodes. In the course of the evaluation, faulty nodes randomly shut down the OSIRIS engine at nodes on which they reside (in the evaluations, for each of the setting, we have 0, 3 or 5 node failures). The most notable observation is that the throughput of the baseline system is lower than the ones of the safety ring configurations. This can be explained with the loss of instances that occurs when a node currently in charge of workflow execution fails and the unavailability of application services during the downtime of a node. Fig. 10 the depicts the number of aborted workflows for the same evaluation runs as in Fig. 9. A workflow is either aborted if (in the non-safety ring case) the node currently executing a workflow instance crashes, or when no application service exists in the network to proceed workflow execution. Note that due to the set-up (no node is able to host more than one service), aborted workflows may even occur in the safety ring configuration, but much less frequently as in the baseline set-up. Naturally, the more faulty nodes there are, the more workflows have to be aborted due to the inherent scarcity of replacement nodes.

When comparing the failure free execution with the ones that are subject to failures, the recovery time can be neglected. Furthermore, the best result can be obtained with a moderate number of safety ring nodes (the throughput of the 5-out-of-11 safety ring node configurations is higher than the throughput of the 11-out-of-11 safety ring node configuration). This can be attributed to the fact that in the case of the 11-out-of-11 safety ring node configuration (i.e., $SRn11s11f0$) safety ring nodes are co-allocated with the computationally intensive activities. These in turn consume significant local resources thus affecting the safety ring node throughput performance.

Similar results are also visible in the 20 node configuration depicted in Fig. 11. Again, the baseline case with 0, 5, and 10 failing nodes is compared against OSIRIS-SR with two configurations, one in which 10-out-of-20 nodes are equipped with the safety ring system service, in the other configuration all 20 nodes are also safety ring nodes. Due to the

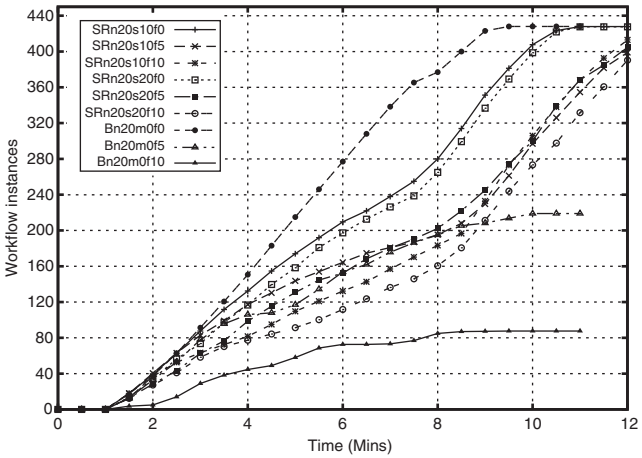

Fig. 11 OSIRIS-SR evaluation, 20 nodes, with failures

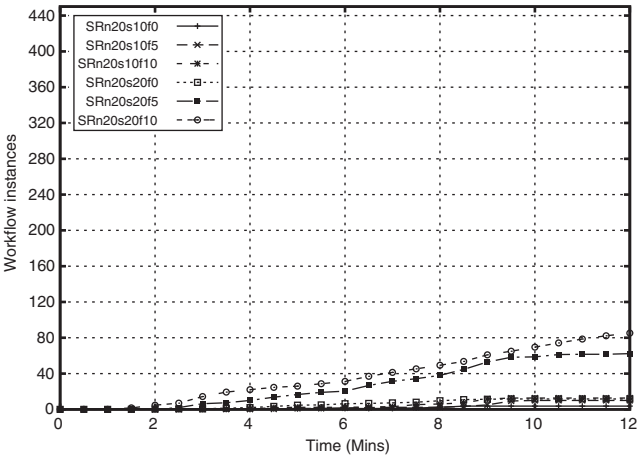

Fig. 12 OSIRIS-SR evaluation, 20 nodes, with failures and aborted workflows

failures and the resulting losses of workflows, the baseline configuration has a rather poor throughput in the presence of failures. On the other hand, the safety ring configurations feature a much higher throughput due to the effective recovery of failed nodes. This can also be observed from Fig. 12 which depicts the number of aborted workflows for the 20 node setting. Given a higher redundancy of application service instances, safety ring can recover the failed nodes by redirecting the workflow execution to substitution nodes and does not have to abort workflows as often as in the case of the 11 node setting (see Fig. 10). Again, not all nodes need to be safety ring nodes, as the configuration with 10-out-of-20 safety ring nodes features an equivalent throughput as the 20 safety ring node configuration. The higher availability of computationally intensive services will balance the load among all nodes, thus offering more local resources for collocated safety ring services.

Overall, the overhead of the safety ring is negligible and does not affect scalability if the system is not affected by node failures. However, if node failures in the system occur, the safety ring manages to recover execution effectively but at the expense of degraded system performance—but with significantly lower workflow abort rates.

7 Related Work

Existing approaches to workflow management such as the IBM WebSphere Application Process Choreographer [14] rely on a centralized database to manage workflow execution metadata. On top of this database, several workflow engines are available for invoking the services that are part of a workflow. While load can be balanced between the workflow engines, the instance database might lead to a performance bottleneck and a single point of failure.

Distributed workflow engines such as Exotica [2] and CEKK [26] provide fault tolerance mechanisms for node failures. Although Exotica/FMQM goes along the same lines as OSIRIS in migrating workflows at runtime, its recovery techniques are based on persistent message queues. These are used to locally store the workflows at migration time. This implies that recovery is only able to handle temporary failures. CEKK takes the persistent message queues a step forward by enforcing them by means of 2PC replication to a set of backup nodes. [25] extends CEKK with scope managers that oversee distributed workflow instance migration by storing backup data. In contrast to the OSIRIS safety ring, these scope managers are centralized nodes that do not scale well.

In [24], redundant service instances are equipped with agents that perform monitoring and recovery. Although this approach does not rely on centralized nodes, it again suffers from potential scalability issues. In the event of active service instance failure, a non-scalable leader election algorithm has to be conducted among the redundant recovery agents. On the contrary, in OSIRIS-SR, the service instance is effectively recovered by a safety ring node, and a safety ring node is recovered, if necessary, by its successor in the safety ring.

[12, 13] propose to use rollback recovery in conjunction with state checkpointing for mobile agents. These approaches rather focus on optimizing the checkpointing frequency for the sake of improved execution performance and do not consider advanced network failure scenarios, i.e., the backup stores are considered to be idle.

A similar approach to achieve reliability in P2P systems is proposed in [1] which considers virtual peers that are composed of multiple redundant ones and that share process state by means of Paxos replication. However, this approach organizes the peers with a network overlay that is not evenly distributed (not subject to load balancing) when high numbers of concurrently running workflows have to be considered.

Distributed workflow management based on smart agents such as NIÑOS [15] or AMOR [4] optimize for the allocation of resources of nodes by avoiding the provisioning of workflow metadata to all nodes, as it is the case in OSIRIS. Although smart agents traverse nodes at runtime just like workflows, they resolve the execution paths dynamically and thus can perform semantic path detection (i.e., AMOR and CEKK) for recovery purposes. In NIÑOS, dedicated management agents are introduced for each instance of the workflow that monitor and recover failures in the process of execution. Although the these workflow management agents are instantiated for each instance, they are also centralized (residing on a single node only).

JOpera [11, 16] also features a distributed workflow engine. To power JOpera, a global workflow state and environment configuration state is maintained at a centralized node and replicated to all redundant engines at runtime in a P2P fashion such that the centrality of the global repository is alleviated. Each workflow engine node locally maintains routing tables which provide it with information workflow instance state that is located at other nodes. This data however needs to be pre-fetched at runtime from the other nodes before the invocation of a local activity can take place. JOpera achieves fault-tolerance by periodically checking the configuration (i.e., routing tables) state among the nodes by pinging the connected nodes. In case of node failures, the required state for invocation of local activities is obtained from the global repository. For this to function, each workflow engine has to write its updates to the workflow instance state both to its connected nodes w.r.t. the local configuration state and the global repository.

8 Conclusions and Outlook

In this paper, we have presented the combination of OSIRIS and its safety ring extension OSIRIS-SR with Compass, an extension to Chord for optimized routing and data access in P2P data management. The combination is a distributed workflow execution platform that provides a high degree of fault tolerance, despite of its distributed execution, and that is particularly tailored to mobile environments (or mixed mobile/stationary environments) where some nodes in a system are mobile and thus have a significantly higher latency than stationary nodes. The evaluations have shown the effectiveness of OSIRIS-SR's failure handling and the gain in efficiency when using Compass over a Chord-based DHT in the presence of mobile nodes.

In our current work, we are applying the OSIRIS-SR and Compass extensions to streaming services, i.e., services that do not follow a discrete request/reply style of interaction but that are able to process potentially unlimited streams of incoming data. This will allow to support new types of workflows that contain a mix of discrete and streaming services.

Acknowledgments This work has been partly funded by the Swiss National Science Foundation, project SOSOA (contract no. CRSI22 127386/1) and supported by an amazon AWS Grant.

References

1. Abe K, Ueda T, Shikano M, Ishibashi H, Matsuura T (2009) Toward fault-tolerant P2P systems: constructing a stable virtual peer from multiple unstable peers. In: Proceedings of the 1st International Conference on Advances in P2P Systems (AP2PS)
2. Alonso G, Mohan C, Gunthör R, Agrawal D, Abbadi AE, Kamath M (1995) Exotica/FMQM: a persistent message-based architecture for distributed workflow management. In: IFIP Information Systems Development for Decentralized Organizations, pp 1–18
3. Alves et al (2007) Web services business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. Accessed 2 March 2015
4. Binder W, Constantinescu I, Faltings B, Haller K, Türker C (2006) A multiagent system for the reliable execution of automatically composed Ad-hoc processes. Autonomous Agents and Multi-Agent Systems 12(2):219–237
5. Brandom R (2013) Fighting fire with data: how mapping tech helped beatback the Rim Fire. http://www.theverge.com/2013/9/21/4753164/fighting-fire-with-maps-how-gis-tech-helped-beat-back-the-rim-fire
6. Brettlecker G, Schuldt H (2011) Reliable Distributed Data Stream Management in Mobile Environments. Inform Syst J 36(3):618–643
7. Brettlecker G, Milano D, Ranaldi P, Schek H-J, Schuldt H, Springmann M (2007) ISIS and OSIRIS: a process-based digital library application on top of a distributed process support middleware. In: Proceedings of the 1st International DELOS Conference on Digital Libraries: Research and Development, volume 4877 of *LNCS*. Springer, Pisa, pp 46–55
8. Cachin C, Guerraoui R, Rodrigues L (2011) Introduction to reliable and secure distributed programming, 2nd ed. Springer, Heidelberg
9. Ghodsi A, Alima LC, Haridi S (2007) Symmetric replication for structured Peer-to-Peer systems. In: Proceedings of the International Conference on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P), DBISP2P'05/06, Springer-Verlag, Trondheim
10. Gray J, Lamport L (2006) Consensus on transaction commit. ACM T Database Syst 31(1):133–160
11. Heinis T, Pautasso C, Alonso G (2005) Design and evaluation of an autonomic workflow engine. In: Proceedings of the 2nd International Conference on Autonomic Computing (ICAC'05), pp 27–38
12. Jafar S, Krings A, Gautier T (2009) Flexible rollback recovery in dynamic Heterogeneous grid computing. IEEE Trans Dependable Secure Comput 6(1):32–44
13. Kaur R, Challa RK, Singh R (2010) Antecedence graph based checkpointing and recovery for mobile agents. In: Proceedings of the International Conference on Communication Control and Computing Technologies (ICCCCT'10), pp 491–424
14. Keen M, Brown B, Garratt A, Käckenmeister B, Khairy A, O'Mahony K, Yu L (2010) Building IBM business process management solutions using WebSphere V7 and business space. Number SG24-7861-00 in IBM Redbooks. IBM. May 2010
15. Li G, Muthusamy V, Jacobsen H-A (2010) A distributed service oriented architecture for business process execution. ACM Transactions on the Web 4(1)

16. Pautasso C, Alonso G (2005) JOpera: a toolkit for efficient visual composition of web services. Technical Report 2, International Journal of Electronic Commerce (IJEC)
17. Schuldt H, Alonso G, Beeri C, Schek H-J (2002) Atomicity and isolation for transactional processes. ACM T Database Syst 27(1):63–116
18. Schuler C, Schuldt H, Türker C, Weber R, Schek H-J (2005) Peer-to-peer execution of (transactional) Processes. Int J Coop Inf Syst 14(4):377–406
19. Schuler C, Türker C, Schek H-J, Weber R, Schuldt H (2006) Scalable Peer-to-Peer process management. International Journal of Business Process Integration and Management (IJBPIM) 1(2):129–142
20. Stoica I, Morris R, Karger D, Kaashoek MS, Balakrishnan H (2001) Chord: a scalable peer-to-peer lookup service for internet applications. In: Proceedings of the ACM conference on applications, technologies, architectures, and protocols for computer communication (SIGCOMM 2001), New York, pp 149–160
21. Stojnić N, Schuldt H (2013) OSIRIS-SR—A scalable yet reliable distributed workflow execution engine. In: Proceedings of the 2nd International Workshop on Scalable Workflow Enactment Engines and Technologies (SWEET'13), New York City
22. Stojnić N, Probst L, Schuldt H (2013) COMPASS—optimized routing for efficient data access in mobile chord-based P2P systems. In: Proceedings of the 14th International Conference on Mobile Data Management (MDM'13), vol. 1. IEEE Computer Society, Milan, pp 46–55
23. Weber R, Schuler C, Neukomm P, Schuldt H, Schek H-J (2003) WebService composition with O'GRAPE and OSIRIS. In: Proceedings of 29th International Conference on Very Large Data Bases (VLDB 2003), Morgan Kaufmann, Berlin, pp 1081–1084
24. Ye X (2006) Towards a Reliable Distributed Web Service Execution Engine. In: Proceedings of the IEEE International Conference on Web Services (ICWS'06), Chicago, pp 595–602
25. Yu W (2010) Fault handling and recovery in decentralized services orchestration. In: Proceedings of the 12th International Conference on Information Integration and Web-based Applications and Services (iiWAS'10), Paris, pp 98–105
26. Yu W, Yang J (2007) Continuation-passing enactment of distributed recoverable workflows. In: Proceedings of the 22nd Annual ACM Symposium on Applied Computing (SAC 2007), Seoul, pp 475–481