A Distributed Monitoring and Reconfiguration Approach for Adaptive Network Computing

Bharat Bhargava, Pelin Angin, Rohit Ranchal
Department of Computer Science
Purdue University
West Lafayette, IN, USA
{bb, pangin, rranchal}@cs.purdue.edu

Sunil Lingayat
Information Systems Sector
Northrop Grumman Corporation
McLean, VA, USA
sunil.lingayat@ngc.com

***Abstract*—The past decade has witnessed immense developments in the field of network computing thanks to the rise of the cloud computing paradigm, which enables shared access to a wealth of computing and storage resources without needing to own them. While cloud computing facilitates on-demand deployment, mobility and collaboration of services, mechanisms for enforcing security and performance constraints when accessing cloud services are still at an immature state. The highly dynamic nature of networks and clouds makes it difficult to guarantee any service level agreements. On the other hand, providing quality of service guarantees to users of mobile and cloud services that involve collaboration of multiple services is contingent on the existence of mechanisms that give accurate performance estimates and security features for each service involved in the composition. In this paper, we propose a distributed service monitoring and dynamic service composition model for network computing, which provides increased resiliency by adapting service configurations and service compositions to various types of changes in context. We also present a greedy dynamic service composition algorithm to reconfigure service orchestrations to meet user-specified performance and security requirements. Experiments with the proposed algorithm and the ease-of-deployment of the proposed model on standard cloud platforms show that it is a promising approach for agile and resilient network computing.**

Keywords—agile computing, resilience, monitoring, adaptability, service-oriented computing

I. INTRODUCTION

Recent advances in services computing including the rise of the cloud computing paradigm has brought network computing, which enables the sharing of computing and storage resources over a network and collaboration of services to achieve complex tasks, to a whole new level. Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [9]. Context plays a very important role in achieving high quality of service with both mobile and cloud computing, as both computing paradigms face highly dynamic conditions, i.e. highly variable context. Richness and efficient gathering/utilization of contextual information gains even more importance in the case of collaboration between mobile and cloud platforms, as achieving high computational performance and user satisfaction is contingent upon the correct identification of resources to be utilized and various constraints peculiar to mobile computing.

Context in network computing consists of the following elements [6][2]:

1. User preference: This element of context captures user-specific preference settings for particular applications, and could potentially affect the composition of mobile services.
2. Workload: This element involves the workload on the mobile services (the usage of resources by different applications, or by different components of an application), and is another highly dynamic element of context.
3. Data connection type, bandwidth: The available bandwidth between clients and services and the data connection type are important factors especially for real-time data-processing intensive applications.
4. Resource availability: This contextual clue captures the availability and quality of various resources on specific hosts, and can be used to make decisions regarding the selection of services to achieve the best application performance.
5. Situational context: This context element consists of monitored data and information regarding the user location, time and other data collected by sensors. It is most useful in personalizing applications to achieve high user satisfaction.

Adaptability to different contexts is significant for high performance in network computing [3]. In order to ensure the enforcement of service level agreements (SLAs) and provide high security assurance in network computing in real time**,** a monitoring framework needs to be developed to inspect the services dynamically during their execution. If a service is compromised, misbehaves or is underperforming, the service monitor needs to *discover this inadequate performance, provide feedback, take remedial actions and adapt according to the changes in context.* The monitoring system must not incur a high performance overhead to make them viable for large-scale deployment. Finally, a good monitoring framework should be as transparent to the service providers and

consumers. The last requirement is essential for adoption in the industry.

There is a need for novel techniques to monitor service activity, discover and report service behavior changes, and enforce security and quality of service requirements in cloud and mobile services [1] by integrating the various elements of context mentioned above. In this paper, we propose a novel approach that uses a distributed network of service activity monitors to audit and detect service behavior and performance changes in various domains. By integrating components for *service performance monitoring, service trust management* and *dynamic service composition,* the proposed model aims to provide a unified architecture for agile and resilient network computing.

The rest of this paper is organized as follows: Section II provides a brief review of related work in service auditing and policy enforcement in network computing. Section III presents our proposed system architecture for agile and resilient computing. Section IV provides the details of dynamic service composition algorithm in the proposed model and Section V presents the results of experiments with the algorithm. Section VI concludes the paper with future work directions.

II. Related Work

Development of runtime-auditing systems for mobile and web-based services has been the focus of many research efforts. Li et al. [7] describe a system for auditing runtime interaction behavior of web services. They use finite state automata to validate predefined interaction constraints, where message interception is bound to the particular server used for deploying Web services. Simmonds et al. [10] present a more comprehensive auditing solution for checking behavioral correctness of web service conversations. They use UML 2.0 Sequence Diagrams as a property specification language, which are then transformed to automata by multiple monitors that check the validity of safety and liveness properties. Their proposal is for a specific application server, since they utilize an event mechanism provided by that server. Guinea et al. [4] present an auditing system that is able to report and monitor functional requirements and quality-of-service constraints of BPEL (Business Process Execution Language) processes. Their approach leverages data collection, including message interception for auditing. However, their solution is technology-dependent and they do not address the auditing of cloud-based service interactions.

To support flexible auditing of the behavior pattern for composite services, Wu et al. [12][13] demonstrate an "aspect extension" to WS-BPEL, in which history-based *pointcuts* specify the pattern of interest within a range, and *advices* describe the associated action to manage the process if the specified pattern occurs. The solution they provide addresses specific orchestration engines, which is not a generic solution for modern cloud-based and mobile services. In [8] and [11] the identification of trusted services and dynamic trust assessment in SOA are studied. Malik et al. [8] introduce a framework called *RATEWeb* for trust-based service selection and composition based on peer feedback. It is based on a set of decentralized techniques for evaluating reputation-based trust with ratings from peers. However they do not take into account initial service invocations and the secondary services in compositions. Spanoudakis et al. [11] present an approach to keep track of trusted services to address the compliance of promises expressed within their service level agreements (SLAs). The trust assessment is based on information collected by monitoring services in different operational contexts and subjective assessments of trust provided by different clients (consumers) situated in specific operational context. They further address the issue of unfair ratings by combining user rating and quality of service monitoring. Approaches like [8] and [11] are not suitable for compositions with many services, because the monitoring system would need to collect intensive information from a lot of peers and clients, which would make it very expensive.

Gamble et al. [14] present a tiered approach to auditing information in the cloud. The approach provides perspectives on auditable events that may include compositions of independently formed audit trails. Filtering and reasoning over the audit trails can manifest potential security vulnerabilities and performance attributes as desired by stakeholders. [5] introduces a system to model the essential security elements and define the proper message structure and content that each service in the composition must have, based on a security meta-language (SML) that models the security relevant portions of the standards for their consistent, comprehensive, and correct application. Both of these approaches focus on how services can comply with established standards, but their implementation requires strong support from the cloud provider (and hence dependency on a certain technology) and extensive changes in the current infrastructures. On the other hand, the solution proposed for agile and resilient monitoring in this paper is generic and can easily be applied to different technologies.

III. Proposed System Architecture

In this section we present our proposed system architecture for the monitoring of services in network computing in order to provide adaptability and resiliency in the case of changing service behavior and context.

The main goals of agile and resilient computing in the proposed approach are:

1. Replacing anomalous/underperforming services with reliable versions
2. Reconfiguring service orchestrations to respond to anomalous service behavior
3. Swiftly self-adapting to changes in context
4. Enforcing proactive and reactive response policies to achieve performance and security goals
5. Achieving continuous availability even under attacks and failures

Providing adaptability in order to achieve increased resiliency relies on two main elements:

1. *Monitoring service status and determining action:* Monitoring of services is of utmost importance in achieving high resilience, as services in environments with highly dynamic contexts such as mobile and

cloud computing platforms may exhibit frequent changes in many quality of service parameters such as throughput, bandwidth, response time etc. Compliance with SLAs is also hard to achieve in such dynamic environments, making monitoring a must for accurate decision-making in service composition problems.

2. *Dynamic reconfiguration of services based on changes in context:* Changes in both service context and the context of users can affect service compositions, requiring dynamic reconfiguration. While changes in user context can result in updated priorities such as trading accuracy for shorter response time in an emergency, as well as updated constraints such as requiring trust levels of all services in a composition to be higher than a particular threshold in a critical mission, changes in service context can result in failures requiring the restart of a whole service composition. Dynamic reconfiguration of service compositions based on the updated constraints and service contexts enables enhanced enforcement of policies to provide higher user satisfaction.

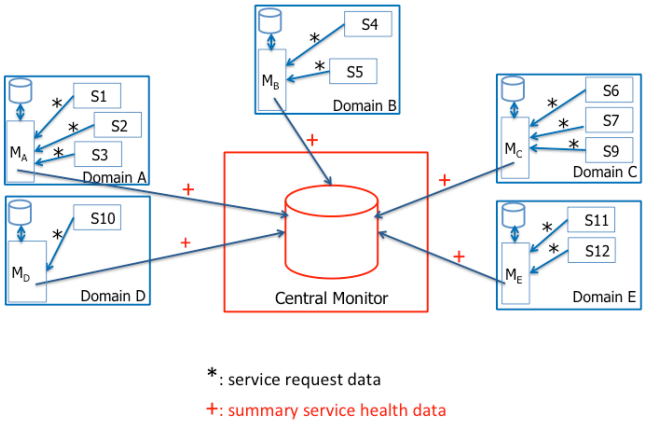

Fig. 1. Broad view of the distributed monitoring system architecture.

Figure 1 shows a high-level view of the proposed system architecture. Monitoring in the proposed model is distributed in the sense that each service domain, such as a cluster of machine instances in the cloud or a set of mobile services in close proximity to each other, has a service monitor that tracks interactions among the services in the domain as well as interactions with services or users outside the domain. When a service is deployed, it is registered with the local monitor of its domain in order to be discoverable by other services or users. The local monitors have access to all interactions with the services registered in their domain and they gather data including response time, response status, authentication failures etc. among other parameters for each service using interceptors transparent to each service implementation, and log them in their database. The data collected are periodically mined by each local monitor using statistical analysis of multi-lateral time-series data to detect deviations from normal behavior and reported to a central monitor in the form of summary health statistics and trust values for the services. These statistics are utilized by the dynamic service composition module described in the following section when making decisions about which services to include in a specific orchestration to meet user requirements.

In the prototype system, each local service monitor is implemented using Apache Axis2 [16] valves for intercepting all service requests in the domain and each service domain includes a MySQL [17] database, in which useful data (response time, response status, CPU usage, memory usage) about each service gathered by the monitor is logged. The central monitor is implemented as a web service on Amazon EC2 [15], which has its own database to store health, endpoint address and category data for various services. While each service invocation leads to an update in the local monitor's database, summary data for all services in a specific domain is reported to the central monitor periodically by each local monitor. The dynamic service composition module, as described in the next section, utilizes information from the central monitor's database to create service orchestrations that comply with users' performance and/or security requirements.

IV. Dynamic Service Composition

A. Dynamic Service Composition Problem

The challenge in dynamic service composition is to configure set of services that conform to security policy and quality of service requirements. A dynamically reconfigured service composition is based on changes in the context with respect to timeliness and accuracy of information as well as the type, duration, extent of attacks and the complexity of the threat environment. Configurability needs rules that allow applications and customers to set priorities, risk tolerance, and monitoring requirements. In network computing, every service orchestration is composed of a series of services that interact with each other based on an *interaction graph*. One of the benefits of cloud computing is that there can be multiple options for services to achieve a specific task. We define a *service category* as an abstraction for a set of services that provide similar functionality. A *service* is the actual implementation of the functionality for a specific service category.

The left hand-side of Figure 2 shows a specific orchestration of services involving categories A, B and C, where service A-1 invokes service B-1, which invokes C-1. When we perform dynamic service composition, we effectively choose services for each category in the composition. For example, if service B-1 in Figure 2 fails, it can be replaced with service B-2, which also results in the replacement of service C-1 with C-3 as shown on the right hand-side of the figure.

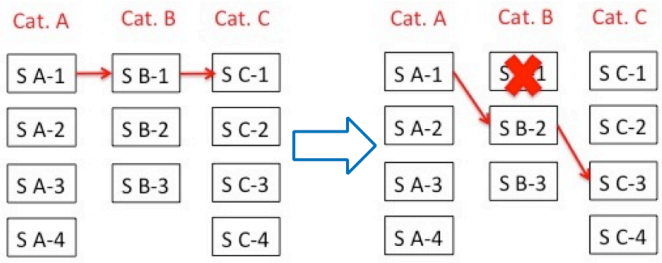

Fig. 2. Example of a configuration change in the case of service failure.

Since there are multiple services in every service category, we face the challenge of selecting the most secure and best performing service orchestration out of the available services. This problem is challenging, as it requires meeting multiple criteria such as security, availability, and cost of a service, etc. The goal of dynamic service composition is to maximize the resiliency and trustworthiness of the system based on selecting the best individual services, while meeting the constraints (security and SLA requirements). Using the central service monitor, we maintain the latest values for the trust and QoS parameters of the services and utilize the values of those parameters when making decisions regarding whether to include a specific service in a composition.

B. Dynamic Service Composition Algorithm

The problem of finding an optimal service composition subject to a set of performance and security constraints is an NP-hard problem. As achieving low response times for dynamic service composition requests is important in real-time computing, we propose a greedy heuristic-based approach to find near-optimal solutions to this problem.

Algorithm 1. Dynamic Service Composition

```
//Input: A set of m service categories S_i, each with a set of
//concrete services.
//Output: A set of services, one in each category, with
//near-optimal utility based on context.

sort services in every category in descending order of utility

for i: 1..m
  x_i1 = 1; // select the highest utility service for the initial
            //composition

if composition satisfies constraints
   return current composition;

else
   while solution not feasible
       //downgrade using the service with biggest
       //aggregate saving in constraints
       for each category i
          for each service j in i
             calculate aggregate saving of service in
                category i of the current composition with j;
          find service j with maximum aggregate saving;
          record the best service BEST(i) for category i (over
                                                   all j's);
      find the overall best service from the set of best
           services for every category (over all i's);
      include the overall best service in the composition;

   return composition;
```

Algorithm 1 provides the details of the proposed solution. Note that each service in the problem has a utility measured by the value of the parameter selected as the target for the optimization problem (i.e. the value we would like to maximize, such as the total trust value of services). Additional service parameters such as response time can be specified as performance/security constraints (e.g. total response time < X).

V. Experiments

We performed experiments to evaluate the response time of the dynamic service composition algorithm for varying number of service categories involved in the composition, as well as varying number of services in each category. The dynamic service composition module was hosted on an Amazon EC2 m3.medium instance (1 vCPU, 3.75 GB memory) in the experiments, where service data was stored in a MySQL database.

The first set of experiments involves dynamic composition requests for orchestrations involving different number of service categories, where each category has 3 services to choose from. More specifically, the task is to create a service orchestration with minimum response time, subject to the constraint that the average trust level of all services in the composition is not below a certain threshold. Figure 3 shows the results of these experiments. As seen in the figure, composition response time does not increase significantly with increasing number of service categories in a composition, and is always under 800 ms, which is a reasonable overhead for most scenarios. Database access time dominates the response time in these experiments.

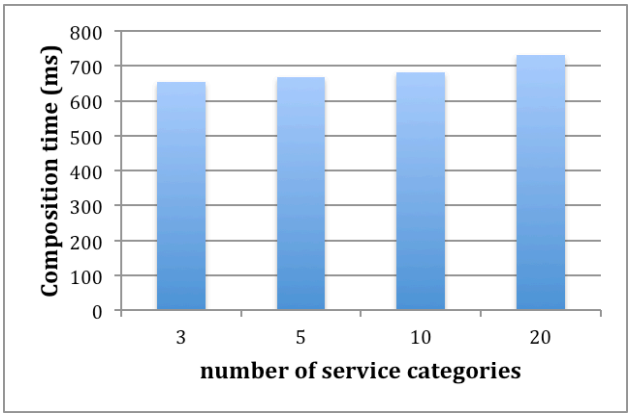

Fig. 3. Dynamic service composition time for varying number of service categories, with 3 possible services for each category.

The second set of experiments involves dynamic composition requests for orchestrations of 3 service categories, with a varying number of services to choose from each category. Figure 4 shows the results of these experiments. As seen in the figure, composition response time does not increase significantly with increasing number of services to choose from for each category, and is always under 800 ms, which is a reasonable overhead for most scenarios.

Once again, database access time is the dominant factor in the response time.

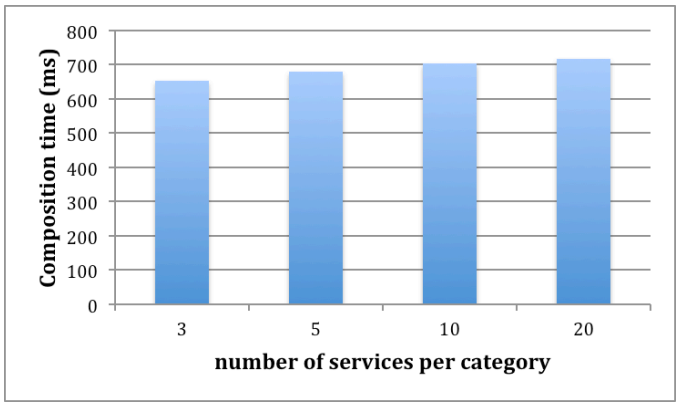

Fig. 4. Dynamic composition time for 3 service categories, with varying number of services for each category.

VI. Conclusion and Future Work

In this paper we proposed an agile and resilient approach for dynamic monitoring and reconfiguration of service orchestrations in network computing. The main impact of the approach is the proposal of a comprehensive monitoring and reconfiguration architecture for network computing involving mobile and cloud services, which achieves high performance and continuous availability even under highly-dynamic contexts involving attacks and service failures, thereby providing increased resiliency. The results of the experiments with the proposed dynamic service composition model and the reliance of the approach on standard technologies make it promising as a preliminary basis for a high-performance distributed architecture in network computing.

Future work will involve comprehensive experiments with the proposed model under highly variable contexts such as fluctuating network bandwidth, changes in service behavior (e.g. CPU/memory utilization patterns), different service loads, and various types of attacks on services that affect performance. We will also evaluate the effects of specifying multiple quality of service and security constraints on the performance of dynamic service composition.

Acknowledgment

This research is supported by Northrop Grumman Cybersecurity Research Consortium.

References

[1] M. Azarmi, B. Bhargava, P. Angin, R. Ranchal, N. Ahmed, A. Sinclair, M. Linderman, and L. ben Othmane, "An End-to-End Security Auditing Approach for Service Oriented Architecture," *Proc. 31st IEEE Symposium on Reliable Distributed System*, 2012, pp. 279-284.

[2] P. Angin, "Autonomous Agent-Based Mobile-Cloud Computing," Ph.D. Thesis, Purdue University, Dec. 2013.

[3] B. Bhargava and S. Browne, "Adaptable Recovery Using Dynamic Quorum Assignments," *Proc. 16th International Conference on Very Large Data Bases (VLDB),* 1990, pp. 231-242.

[4] L. Baresi, S. Guinea, and O. Nano, "Comprehensive Monitoring of BPEL Processes," *IEEE Internet Computing*, Vol. 14, No. 3, 2010, pp. 50-57.

[5] R. Baird and R. Gamble, "Developing a Security Meta-Language Framework," *Proc. 44th Hawaii International Conference on System Sciences*, 2011, pp. 1-10.

[6] H. J. La and S. D. Kim, "A Conceptual Framework for Provisioning Context-Aware Mobile Cloud Services," *Proc. 3rd IEEE International Conference on Cloud Computing (CLOUD'10)*, 2010, pp. 466-473.

[7] Z. Li, Y. Jin, and J. Han "A Runtime Monitoring and Validation Framework for Web Service Interactions," *Proc. Australian Software Engineering Conference*, 2006, pp. 70–79.

[8] Z. Malik and A. Bouguettaya, "Rateweb: Reputation Assessment for Trust Establishment among Web Services," *VLDB*, Vol. 18, No. 4, 2009, pp. 885–911.

[9] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST Special Publication 800-145, 2011.

[10] J. Simmonds, Y. Gan, M. Chechik, S. Nejati, B. O'Farrell, E. Litani, and J. Waterhouse, "Runtime Monitoring of Web Service Conversations," *IEEE Transactions on Service Computing,* Vol. 2, No. 3, 2009, pp. 223-244.

[11] G. Spanoudakis and S. LoPresti, "Web Service Trust: Towards a Dynamic Assessment Framework," *Proc. IEEE International Conference on Availability, Reliability and Security* (*ARES'09*), 2009, pp. 33–40.

[12] G. Wu, J. Wei, and T. Huang, "Flexible Pattern Monitoring for WS-BPEL Through Stateful Aspect Extension," *Proc. IEEE International Conference on Web Services (ICWS '08)*, 2008, pp. 577 – 584.

[13] G. Wu, J. Wei, C. Ye, H. Zhong, and T. Huang, "Detecting Data Inconsistency Failure of Composite Web Services through Parametric Stateful Aspect," *Proc. IEEE International Conference on Web Services*, 2010, pp. 68-75.

[14] R. Xie and R. Gamble, "A Tiered Strategy for Auditing in the Cloud," *Proc. IEEE 5th International Conference on Cloud Computing* (*CLOUD*), 2012, pp. 945-946.

[15] Amazon EC2. http://aws.amazon.com/ec2. Accessed: 14 July 2015.

[16] Apache Axis2. http://axis.apache.org/axis2/java/core. Accessed: 14 July 2015.

[17] MySQL. http://www.mysql.com. Accessed: 14 July 2015.