# GEMINI: A Hybrid Byzantine Fault Tolerant Protocol for Reliable Composite Web Services Orchestrated Delivery

Islam Elgedawy

***Abstract*—To ensure reliable delivery for composite web services, we argue that Byzantine fault tolerance (BFT) must be guaranteed for the service delivery modules (such as BPEL execution engines and dispatchers) as well as for the realizing components of the composite web service. Existing BFT service delivery approaches are mainly focused on atomic web services. However, there are few approaches discussed BFT for composite web services delivery. Unfortunately, such approaches either ensured BFT for the service delivery modules alone or for the service realizing components alone; but not for both. To overcome such limitation, this paper proposes GEMINI; a hybrid BFT protocol for reliable composite web services orchestrated delivery. GEMINI uses a light-weight replication-based BFT protocol to ensure the BFT for service delivery modules, and uses a speculative quorum-based BFT protocol to ensure components BFT. Unlike existing quorum-based BFT approaches that ensure components redundancy via component replication; GEMINI ensures components redundancy via components parallel provisioning. Experimental results show that GEMINI increases the reliability and throughput of composite web service delivery when compared with existing composite web services delivery approaches.**

***Index Terms*—Byzantine faults, composite web services, GEMINI, service delivery.**

## I. Introduction

A composite web service realizes a given business workflow (i.e., describing a given business process) by invoking different web services (known as component services, or simply components). Usually, such components interact with each other via an orchestrator or a dispatcher in order to achieve the required business objectives. To ensure the quality of the composite web service, customers jointly with the composite service provider define the required service level agreement (SLA). Composite web services providers do their best to fulfill their SLAs in order to avoid penalties, and to increase customers' satisfaction. Indeed, this is not an easy task as it requires automated SLA management for composite web services. Automated SLA management requires an advanced full-fledged service delivery system that is capable of handling many complex functionalities such as automated capacity management, automated components coordination and execution, also it should support tasks recovery as well as components discovery, and nevertheless it should be able to support automated cancellation and billing management. To handle all such complex functionalities, we previously proposed CRESCENT [1]; a service delivery framework for composite web services; depicted in Fig. 1. However, this is not enough to ensure reliable composite web services delivery, as the service realizing components as well as CRESCENT modules themselves are unreliable by nature. For example, a service component might behave arbitrarily and deviates from its expected specification; leading to a component failure. Component failure could be resulting due to physical failure (i.e. the simplest forms of failure), SLA violation (i.e. reported from components monitor), and error in computation (i.e. the worst failure type, due to its high costs for detection and correction). In other words, a composite web service could fail due to the Byzantine failures of its components [2]. Same argument also applies over any service delivery module such as the modules of the CRESCENT framework.

In this paper, we argue that in order to achieve reliable composite web service delivery, we have to ensure the BFT for the service delivery modules as well as for the components realizing the composite web service. Unfortunately, most of existing work for BFT service delivery is focusing on atomic web services such as the works in [3]-[7], which cannot be adopted for composite web services as it lacks important tasks such as coordination, components error recovery and component fault isolation [8]. To overcome such limitations, some works appeared to address the issue of composite web services fault tolerance delivery such as the works in [8]-[10]. For example, the work in [9] focused on coordinators BFT and ignored the components fault tolerance. On the other hand, work [8], [10] focused on components fault tolerance and ignored the coordinators. As we can see, none of the existing approaches can guarantee the BFT for the delivery modules and the realizing components.

This paper identifies this gap and proposes GEMINI; a hybrid asynchronous BFT protocol for reliable composite web services orchestrated delivery. GEMINI decouples composite web service logic (i.e. abstract workflows) from its realization (i.e. components), as it supports dynamic components provisioning. GEMINI uses a light-weight replication-based BFT protocol (such as PBFT protocol [11]) to ensure the reliability of service delivery modules, and uses a speculative quorum-based BFT protocol (such as Zyzzyva [12]) to ensure realizing components delivery reliability. However, unlike existing quorum-based BFT protocols that achieves components redundancy via replication, GEMINI achieves components redundancy via parallel provisioning (i.e. invoking different multiple components at the same time to realize a given workflow task). GEMINI also optimizes the current PBFT protocol; by adopting a single leader view.

Manuscript received January 7, 2015; revised May 27, 2015.

Islam Elgedawy is with the Computer Engineering Department, Middle East Technical University, Northern Cyprus Campus, Guzelyurt, Mersin 10, Turkey (e-mail: Elgedawy@metu.edu.tr).

DOI: 10.7763/IJCTE.2016.V8.1071

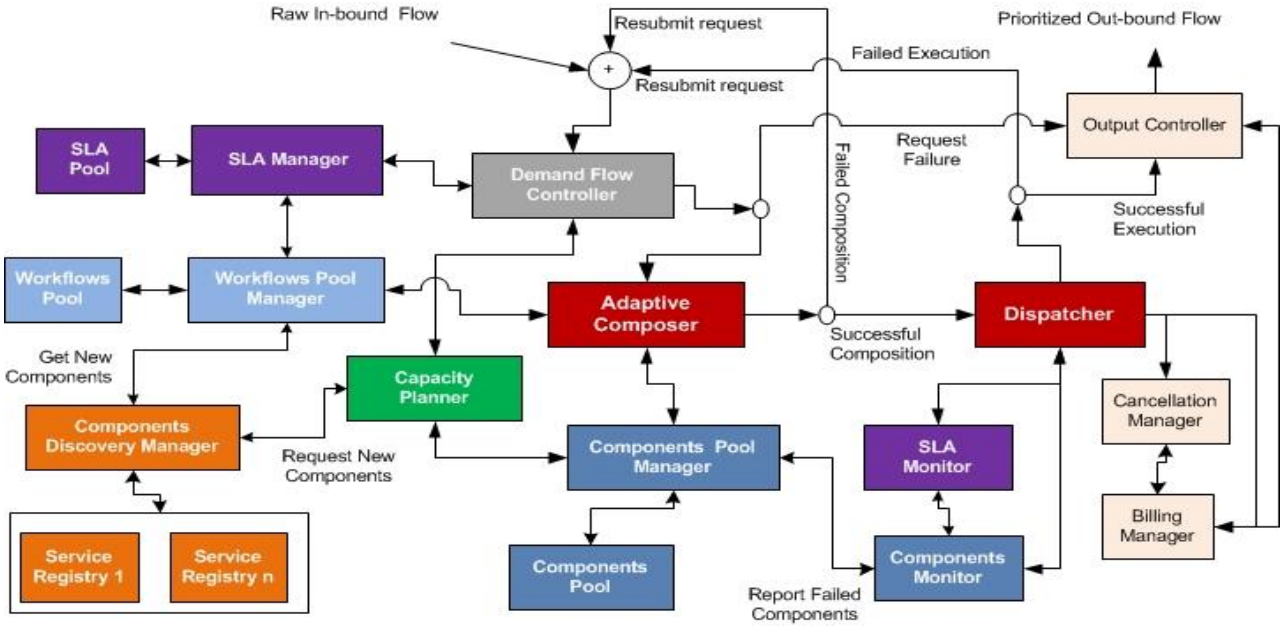


Fig. 1. The CRESCENT framework adopted from [1].

Experimental results show that GEMINI increases the reliability and throughput of composite web service delivery when compared with existing composite web services delivery approaches.

The rest of the paper is organized as follows. Section II provides some background information regarding existing BFT protocols. Section III proposes the GEMINI protocol. Section IV presents the verifying simulation experiments, Section V discusses the related work, and finally, Section VI concludes the paper and proposes directions for future work.

## II. Background

This section provides some background information regarding existing Byzantine fault-tolerance protocols. The key idea behind Byzantine fault tolerance is redundancy. One way to achieve redundancy is via replication, in which multiple replicas of the component are created (forming a cluster of replicas), where clients (i.e. initiators and dispatchers) submit their requests to all replicas or to a primary replica that propagate the requests to all other replicas. Based on the response of all replicas the final response is determined by the BFT protocol. Generally, there are two main approaches for providing Byzantine-fault tolerance: state-machine replication approach (such as the protocols discussed in [2], [11], [13]) and quorum-based approach (such as Zyzzyva [12] and Q/U [14] protocols). We summarize them as follows:

- **The State-Machine Replication BFT Approach:** In this approach, all replicas must communicate with each other to agree on a total order for incoming requests such that each replica execute the incoming requests in the same order. Hence, all replicas must be synchronized in order to guarantee their correctness. However, such communication overhead is not needed when there is no contention periods. Old BFT approaches (such as [2], [13]) adopted synchronous communication between replicas, which is not practical when latency between replicas is high, as in internet-based applications. To overcome such problem, work in [11] proposed a Practical BFT (PBFT) protocol that supports asynchronous communication between replicas, and uses a three-phase commit protocol (i.e. pre-prepare, prepare, and commit phases) to propagate and commit requests to replicas, then requires all replicas to submit their response to the client (the initiator). PBFT requires the client to use a voting system to select the majority response. Hence, the client has to wait for ($f$ + 1) common responses to accept their results as the correct one, where f is the number of supported Byzantine faults. PBFT also tries to minimize the communication overhead between replicas by adopting different view-change policies such as changing only the primary replica rather than changing all replicas in a view when forming a new view. The PBFT protocol proposed by [11] is widely adopted by many researchers as indicated in the related work section.
- **The Quorum-Based BFT Approach**: This approach does not require replicas to agree on a total order and requires clients to contact replicas directly to optimistically execute operations, and uses a voting system to select the majority response as the correct response that the client has to wait for (3$f$ + 1) common responses to accept their results as the correct one. The quorum-based approach requires only one phase for read operations, and two phases for write operations. Hence, it requires much less communication overhead when compared with state-machine replication approach, as it uses the client to detect and correct faulty replicas.

However, both BFT approaches have their shortcomings, the state-machine replication is not scalable due to the inter-replica communication required for determining the total order. Such communication overhead also has a negative impact on the service throughput. On the other hand, the quorum-based approach cost is higher as it requires a large number of replicas: (5$f$ + 1) are needed to tolerate $f$ failures, which is considerably higher than the theoretical minimum of (3$f$ + 1). This increase in the replica set size not only affects the cost but also increases the communication complexity. To

overcome such problems, a new HQ approach [15] is proposed to compromise between these approaches that in non-contention periods it uses the quorum-based approach, while in the contention periods it uses the PBFT protocol. Both PBFT and quorum-based approaches do not support client isolation. Hence, the client is tightly coupled to these protocols. GEMINI overcome such problems by decoupling the clients from an execution details.

## III. GEMINI: A Hybrid Byzantine Fault Tolerant Service Delivery Protocol

GEMINI defines a composite web service as a set of different workflows, defined by any workflow language (such as BPEL). Each workflow task is realized by invoking a set of parallel components, forming what is known as a components cluster. Such components are discovered and adapted using suitable service discovery and adaptation approaches such as the approaches discussed in [16], [17]. Hence, GEMINI requires a delivery framework that minimally consists of a composer, a dispatcher and a component discovery module that finds the required components on the fly from existing components repositories. Of course, other delivery modules could be used as we indicated before in the CRESCENT framework [1].

To ensure BFT for composite web service delivery, we argue that we have to ensure the BFT for the GEMINI delivery modules as well as for components realizing workflow tasks. If we followed a pure BFT replication approach, all GEMINI modules as well as workflow realizing components have to be replicated, and synchronized, which is not a practical approach, due to the very high communication overhead between all replicas; leading to bad performance. Also if we followed the quorum-based approach, a higher number of replicas is required. Furthermore, we cannot replicate the realizing components, as they are usually external services. Hence, we argue that we should combine between practical and quorum-based BFT approaches to minimize such communication overhead, and avoid components replication. We propose to ensure the BFT for delivery modules (such as composers and dispatchers) using a practical replication-based approach (i.e. a light-weight version of PBFT protocol), and ensure the BFT for the workflow tasks delivery using a quorum-based approach (such as Zyzzyva [12]). However, to achieve redundancy required by quorum-based approach, we adopted components parallel provisioning rather than components replication. Adopting parallel provisioning is a great strategy to avoid the high cost of replicas management and synchronization. Furthermore, components realizing workflows are third-party components provided by different service providers, hence replicating such components is not practical and almost not feasible as it requires extensive deployment knowledge about the consumed service providers' resources, which usually is not declared by service providers, and not easy to create. The greatest benefit from adopting parallel provisioning is the ability to increase the number of provisioned components to handle incoming demand spikes, which ensures good performance for the composite web service when such demand fluctuations occur. In what follows we will describe such approach, first we will discuss our approach for quorum-based component BFT, and then we discuss the adopted light-weight version of PBFT approach used for GEMINI modules.

### A. A Quorum-Based BFT Approach for Workflow Task Delivery

In order to ensure BFT for workflow tasks realization, we have to ensure components redundancy (i.e. required by BFT protocols). The GEMINI achieves such redundancy via component parallel provisioning rather than via component replication. Hence, requests will be submitted to a components cluster(s), and the majority response of such cluster will be accepted as the correct answer. Redundancy via provisioning does not require replicas synchronization or management, hence communication overhead is heavily minimized, as we will need only one phase for read and write operations. Fig. 2 shows message interactions of request submission and response collection phases.

We adopt the speculation principle discussed in Zyzzyva to ensure task delivery BFT, hence we need at least $3f + 1$ components in the components cluster. Such number is guaranteed to exist, as GEMINI adaptive composer submits requests to the dispatcher only when parallel composition plan is successfully constructed with component clusters of $3f + 1$ components. The dispatcher in GEMINI will be the initiator for the quorum-based protocol, it has to wait for a matching $3f + 1$ responses to accept the response (see Fig. 2).

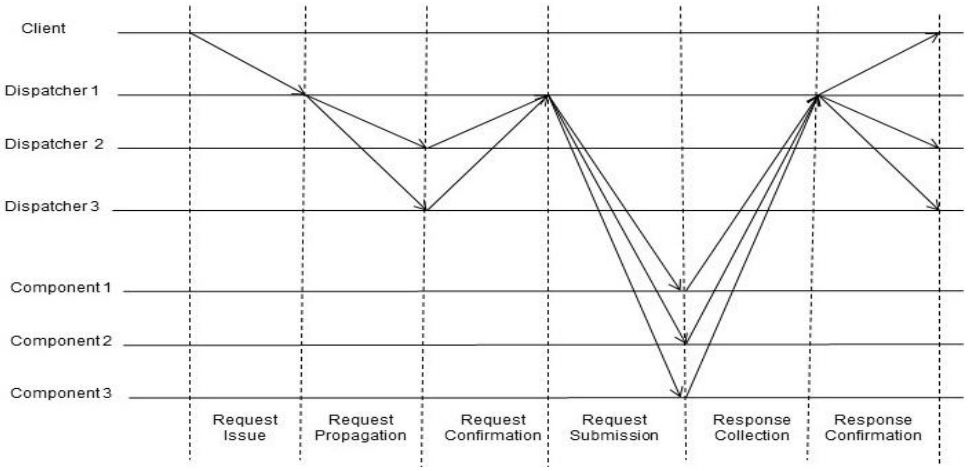


Fig. 2. GEMINI BFT protocol in normal case.

If it received a number between $2f + 1$ and $3f + 1$, it requires commit certificates from components as in Zyzzyva, if less than $2f + 1$ responses are received, this means the response is compromised. Therefore, GEMINI accepts this case as failure and starts a failure recovery protocol to ensure composite web service functional correctness. Error recovery is started by the dispatcher, as it resubmit the request along with a list of compromised components back to the composer in order to find an alternative composition plan with no compromised components. Of course as in Zyzzyva, we assume that every request has a unique number, and each component keeps a history of processed requests such that it checks incoming requests against history first that if a duplicate request is detected, the corresponding stored result is returned to the dispatcher without the need to process the request again (i.e., ensuring the idempotency property).

### B. A Light-Weight PBFT Approach for Delivery Modules

Failure of any of GEMINI modules (such as composers and dispatchers) will jeopardize the whole service delivery process, hence we have to guarantee the BFT for GEMINI delivery modules as well. This is could be easily done via any practical BFT protocol (such as PBFT protocol [11]). However, such protocols requires client involvement to accomplish the required response voting task. We argue that the service delivery process should be totally transparent to the client, such that the client design should be independent from the adopted BFT protocol (i.e. client isolation). Furthermore, these protocols requires replicas synchronization, hence each module replica has to communicate with other modules replicas to process the request, which creates massive communication overhead that degrades the service performance. For example, if a cluster of dispatchers needs to communicate with a cluster of components, this requires every dispatcher replica to invoke each component in the component cluster, which is very expensive. For these two reasons, we did not apply PBFT protocols strictly, however we did some optimizations to minimize such communication overhead, and to isolate the client from the adopted BFT protocol.

We achieved such objectives by adopting the concept of a view and a primary (i.e. a leader) discussed in Paxos [18]. A view in Paxos is a collection of replicas, such replicas elect one replica to be the leader or the primary. All communication should be done via the leader, and the leader should keep other replicas up to date. Each GEMINI module view will have a leader that communicates with other views leaders. For example, the leader of the dispatcher view, informs other dispatcher replicas about the requests being submitted and their corresponding composition plans, also informs them with the requests that have been processed and confirmed by the component cluster. The actual client isolation is guaranteed, as the client will only contact the primary replica of the demand flow controller module, and get the output via the output controller. Fig. 2 depicts the proposed GEMINI BFT protocol in normal case. Fig. 2 shows that once the client issues the request to the primary dispatcher (to explain the protocol in a simple manner, we just omitted the interaction between client and composer, composer and dispatcher, and assumed direct contact between the client and the dispatcher). The primary dispatcher propagates the request to all dispatcher replicas (i.e. request propagation phase), then waits for majority confirmation from other replicas (i.e., request confirmation phase) before it submits the request to the components (i.e. request submission phase). This request confirmation phase is very important as in case of leader failure, other dispatcher replicas can identify uncommitted requests. Once the leader dispatcher submits the request to the components, it follows the quorum-based BFT protocol discussed before, and collect components response (i.e. response collection phase), if the majority of components provided the same response, it accepts the response and sends it to the client and other dispatcher replicas to update their logs (i.e. response confirmation phase).

In case of any dispatcher replica failure, the leader of the view issues a view change to get such replica replaced. The proposed protocol ensures the delivery system liveness, as each replica periodically checks the liveness of its view, and any identified failed replicas including the primary, will be replaced via the view change operations. As we have only one primary at a time, hence only one global order will be enforced through all requests. In case of leader or primary failure, a new leader election process is started.

## IV. Experiments

In this section, we provide simulation experiments conducted to verify basic GEMINI concepts and to compare GEMINI against existing approaches for composite web services delivery. We basically compare GEMINI against the following three approaches. The first approach does not ensure BFT neither for the composite web service components not for delivery architecture modules, as in the approaches discussed in [8] and FACTS [10]. The second approach ensures BFT for the composite web service components but not for the delivery architecture modules, as if the composite web service coordinator adopted the BASE approach [5] for ensuring components BFT. The third approach ensures BFT for the delivery architecture modules but not for the composite web service components, as in approach discussed in [9]. While, GEMINI ensures the BFT for both the components and delivery architecture modules. We compare these approaches using delivery success probability and throughput. Delivery success probability is an indicator for the reliability of the composite web service delivery. That higher success probability implies having more reliable composite web service delivery process. Throughput is the number of requests processed per minute. That higher throughput implies better performance for the composite web service delivery process. To be able to compare the approaches, the adopted performance model is constituted as two main queueing systems in tandem: the dispatcher queueing system and the components queuing system. Any queueing modelling simulation tool could be used to resolve the approaches' queue models once their proper configuration parameters are given. In our experiments, we used the Matlab SimEvents simulator, which provided us with the average response time for each queueing system, and we computed the throughput as the reciprocal of the sum of both average

response times and network latency.

### A. GEMINI Reliability Comparison Experiment

In this section, we will discuss the experiments conducted to compare GEMINI reliability against reliability of existing approaches discussed above. As network latency is the main bottleneck affecting response times on the Internet [19], [20], we will simulate our experiments with high network latency values compared to components processing times. Choice of latency values and processing times is arbitrary as long as such constraint applies. We assumed that there exists 25 msec latency between users and the composite web service delivery system, and 10 msec latency between the delivery system and the composite web service realizing components, also we assumed 5 msec processing time for all composite web service components. We used such values with all approaches in order to have comparable results. The composite service design is arbitrary for our experiments, as ensuring BFT for delivery architecture modules and component parallel provisioning are independent from the service design. Hence, we generated a simple composite web service with a workflow of three sequential tasks. Also we generate demand for the composite service following a Poisson distribution with an arrival rate 60 requests per minute, and we run the simulation for the period of 24 hours. We adopted a redundancy degree of 4 during creation of clusters, which ensures BFT against one Byzantine fault. This degree will be adopted in component provisioning such that each component cluster will contain 4 components, also it will be used when applying PBFT for delivery modules such that each module will have 4 replicas. Of course, adopting higher redundancy degrees increases the cluster reliability, however it degrades its performance [11], [12]. Hence, we choose to go for an average redundancy degree of 4. We computed the success rate for each approach against the expected failure probability, this failure probability is applied for both components and architecture modules. Fig. 3 depicts the obtained results.

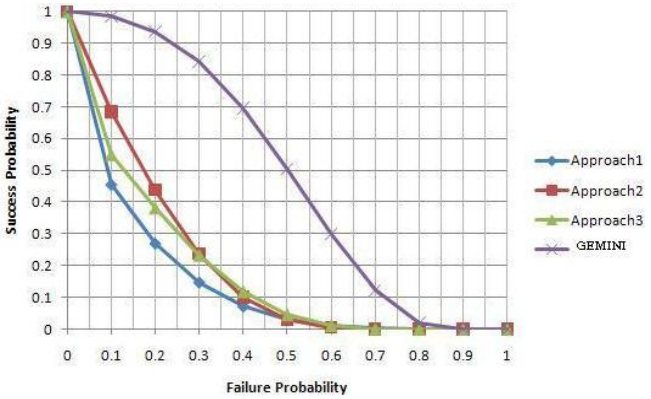


Fig. 3. GEMINI reliability comparison experiment.

Fig. 3 shows that GEMINI increases the delivery success rate when adopted compared with other approaches, as it ensures the BFT for the delivery modules as well as the realizing components. As we can see in results, ensuring BFT only for components (i.e. approach2) or only for delivery modules (i.e. approach3) still provide low success rate, not giving much difference from approach1. Based on the results shown in Fig. 2, we can say to ensure the composite web service reliability, we have to ensure BFT for both of the delivery system modules as well as tasks realizing components. Focusing only on one aspect did not improve the total reliability service delivery by much, however when we took both aspects into consideration (as in GEMINI) significant improvement for total service reliability is achieved. Hence, we argue that adopting GEMINI increases the reliability of composite web services.

### B. GEMINI Performance Comparison Experiments

In this section, we will compare between GEMINI and other approaches in terms of their performance (i.e. throughput). First, we run our simulation with parameters indicated before, however, we assumed no spikes occurs during demand generation, that we have a smooth demand with average arrival rate of 60 requests per minute. In such normal mode, we expect approach1 (i.e. no BFT for modules and components) to have the highest throughput all over other approaches, as there is no time wasted in communication overheads required for replicas synchronization required in BFT. However, GEMINI has a nice feature, that it can distribute incoming demand over multiple component clusters via generating different composition plans to handle incoming requests. Hence, effect of communication overhead wasted to ensure BFT for GEMINI modules could be minimized by the gains obtained from distributing the demand over multiple clusters. In what follows, we will show the GEMINI performance in different scenarios:

1) **Normal Scenario-Smooth Demand:** To show such scenario, we performed the simulation experiment described before with a new parameter Max-Components-per-Cluster, which is max number of components clusters to be allocated for a given workflow task. We repeated the experiments by setting Max-Components-per-Cluster to be 1, 2, 4, 8, and 10. We compute the throughput of all approaches for every case. Fig. 4 depicts the obtained results.
2) **Contention Scenario 1: Demand Spikes with Enough Components:** To show such scenario, we applied our experiment as before with Max-Components-per-Cluster set for 10 component clusters (i.e. we have components abundance), then generated demand spikes with arrival rates of 100, 200, 300, 400, and 500 request per minute, for a period of one hour, then we computed the throughput for all approaches in every case. Fig. 5 depicts the obtained results.
3) **Contention Scenario 2: Demand Spikes with Components shortage:** In case there are no enough components discovered to handle incoming demand, GEMINI will not be able to maintain its good performance due to components shortage. To show such effect, we repeated the previous experiments only with Max-Components-per-Cluster set for two. Fig. 6 depicts the obtained results.

Fig. 4 shows that when having only one component cluster, Approach1 has the best performance as expected due to lack of communication overhead. Also we can see GEMINI still performed better than approach2 and approach3. This is because, GEMINI uses component provisioning rather than

component replication (as in approach2), which requires no replica synchronization. Also GEMINI uses a light-weight BFT for its modules rather than PBFT approach (as in approach3), which requires less number of messages.

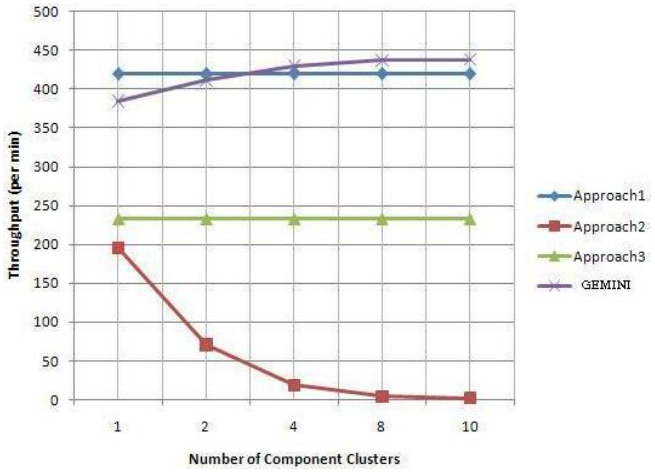


Fig. 4. GEMINI performance comparison experiment-normal case.

However, when we increase the number of components in the pool, and in turn the number of components clusters is increased, the story has changed. As increasing the number of components clusters has no effect on approah1 (i.e. as it adopts no redundancy at all) nor on approach3 (i.e. as redundancy is only applied for delivery modules), we can see their throughput has not changed. However, such increase has very bad effect on approach2, as the communication overhead between components dramatically increased, which can be seen on approach2 performance degradation. On the other hand, GEMINI maintained its good performance, and even better it provided better performance than approach1 when four or more component clusters are used. This is because components have shorter queues due to use of multiple component clusters, hence waiting time is improved, which improves the total response time of the composite service, which increases the overall throughput.

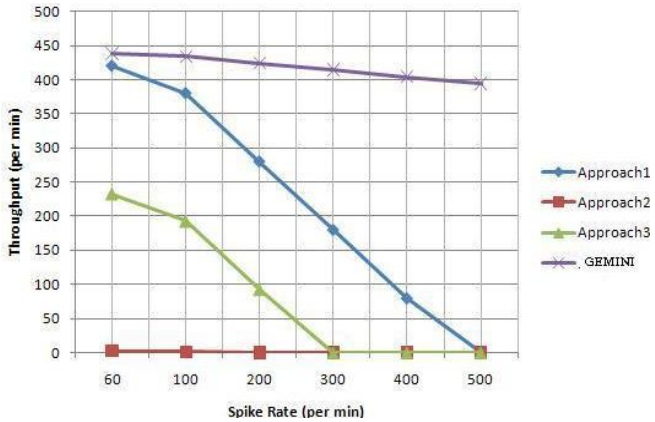


Fig. 5. GEMINI performance comparison experiment-scenario 1.

Fig. 5 shows that GEMINI managed to maintain a good performance, even with very high spikes as 500 request per minute. On the other hand, we can see performance of approach1 and approach3 degraded so badly due to the long queues, even service has failed when spikes are much higher than normal expected rate, we can see approach3 has failed when spike is 300 request per minute, and approach1 failed when spike is 500 requests per minute. Approach2 did not take advantage of such components abundance, as communication overhead between components is very high to the degree it fails the service with the slightest spike increase. Fig. 5 shows that GEMINI managed to survive such high spikes due to its adaptive composition. Hence, as long as we have enough components capable of handling forecasted demand, we believe GEMINI could increase the composite web service performance.

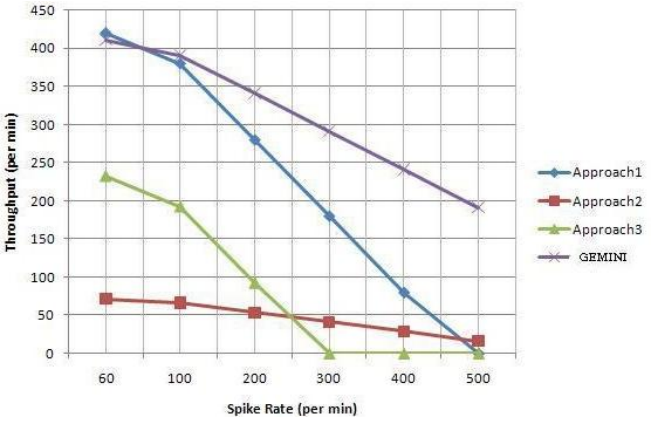


Fig. 6. GEMINI performance comparison experiment-scenario 2.

Fig. 6 shows that as GEMINI could not find enough components to meet incoming demand, hence its performance degraded due to the long queue formed in the front of the existing component clusters. Also we notice performance of approach2 is improved compared to results in Fig, 5, as communication overhead is minimized due to the small number of used components. Even though GEMINI performance has degraded, GEMINI managed to provide better performance than other approaches. Based on the results shown in Fig. 4, 5, and 6, we can say in composite web service delivery, ensuring BFT for workflow tasks via component replication is not recommended due to its bad performance (i.e. as in approach2), however dynamic component parallel provisioning should be used instead (as in GEMINI). Also requiring total synchronization between dispatcher replicas (as in approach3) is not recommended due to its bad performance (i.e. as in approach3), however a lightweight PBFT approach with single leader should be adopted instead (as in GEMINI). Liveness and safety of GEMINI are guaranteed as it already uses existing proven BFT protocols.

## V. Related Work

This section discusses in more details some of the main existing efforts in the areas of BFT web services, and compares between these approaches and GEMINI. Majority of existing research efforts (such as BFT-WS [3], PERPETUAL [4], BASE [5], and THEMA [6]) are focused on ensuring BFT delivery for atomic web services. All these approaches adopted the PBFT protocol [11]. However, there are a few approaches (such as work in [8], [9], FACTS [10]) addressed the composite web service delivery. Work in [9] addressed the issue of BFT coordination for composite web services. The approach is based on the WSBA standard that

simply requires a delivery framework of an initiator, coordinator, and service components. The approach only replicates the coordinator using the PBFT protocol, while the initiator and components remains un-replicated, hence the composite service delivery process could simply fail if one of its components has failed. While work in [10], proposed the FACTS framework for specifying, verifying, and executing fault tolerant composite web services. In FACTS, the service designer can specify different strategies for error handling, one of them is component replication, and also designer could specify the number of retries. These strategies are verified and implemented at run time. Designer could specify alternatives for failed components as a reactive approach. In case of replication strategy is chosen, FACTS uses simple form of parallel provisioning, as it does not wait for majority voting, it simply accepts first reply. Furthermore, if any module of FACTS becomes faulty, the whole composite service delivery process is compromised, as FACTS itself is not fault tolerant, hence FACTS does not ensure BFT composite service delivery. Work in [8] does not support BFT for neither for the delivery framework modules nor for components. It does not replicate clients or components. It requires the client to submit its requests to a dispatcher, which will execute the composite web service components one by one. However, it uses components serial provisioning for handling components failure such that if a given component has failed, another one is tried sequentially, it is more like applying alternatives strategy. None of all above approaches support BFT for both of delivery modules and the realizing components. While, GEMINI overcomes this limitation by combining between quorum-based and replication BFT protocols as shown before.

## VI. Conclusion and Future Work

A In this paper, we argued that reliable composite web service delivery requires ensuring the BFT for the delivery modules as well as for the realizing components. Therefore, we proposed GEMINI; a hybrid asynchronous Byzantine fault tolerant protocol for reliable composite web services orchestrated delivery. GEMINI ensured the BFT for the delivery process by combining between quorum-based and practical BFT approaches. It uses optimized single leader practical BFT approaches to ensure BFT for delivery modules, while it uses quorum-based approaches for ensuring BFT for the realizing components, where redundancy is achieved via provisioning rather than replication. Experimental results showed that GEMINI increases the reliability and throughput of composite web service delivery when compared with existing approaches. Future work will mainly focus on GEMINI cloud deployment.

## References


[1] I. Elgedawy, "CRESCENT: A reliable framework for durable composite web services management," *The Computer Journal*, vol. 58, no. 2, pp. 280-299, February 2015.

[2] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382-401, 1982.

[3] W. Zhao, "BFT-WSB: A byzantine fault tolerance framework for web services," presented at the Middleware for Web Services Workshop, 2007.

[4] S. L. Pallemulle, H. D. Thorvaldsson, and K. J. Goldman, "Byzantine fault-tolerant web services for n-tier and service oriented architectures," presented at the 28th IEEE International Conference on Distributed Computing Systems (ICDCS), 2008.

[5] M. Castro, R. Rodrigues, and B. Liskov, "BASE: Using abstraction to improve fault tolerance," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 236-269, 2003.

[6] M. G. Merideth, A. Iyengar, T. Mikalsen, I. Rouvellou, and P. Narasimhan, "Thema: Byzantine-fault-tolerant middleware for web services applications," presented at the 24th IEEE Symposium on Reliable Distributed Systems (SRDS), 2005.

[7] N. Looker, M. Munro, and J. Xu, "Increasing web service dependability through consensus voting," presented at the 29th Annual International Conference on Computer Software and Applications Conference, 2005.

[8] V. O. Onditi, G. Dobson, J. Hutchinson, J. Walkerdine, and P. Sawyer, "Specifying and constructing a fault-tolerant composite service," presented at the IEEE Sixth European Conference on Web Services, 2008.

[9] W. Zhao and H. Zhang, "Byzantine fault tolerant coordination for web services business activities," presented at 2008 IEEE International Conference on Services Computing, 2008.

[10] A. Liu, Q. Li, L. Huang, and M. Xiao, "Facts: A framework for fault-tolerant composition of transactional web services," *IEEE Transactions on Services Computing*, vol. 3, no. 1, pp. 46-59, 2010.

[11] M. Castro and B. Liskov, "Practical byzantine fault tolerance," presented at the Third Symposium on Operating Systems Design and Implementation, 1999.

[12] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," *ACM Trans. Comput. Syst.*, vol. 27, no. 4, January 2010.

[13] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 299-319, December 1990.

[14] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. Wylie, "Fault-scalable byzantine fault tolerant services," *SIGOPS Oper. Syst. Rev.*, vol. 39, no. 5, pp. 59-74, October 2005.

[15] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira, "HQ replication: A hybrid quorum protocol for byzantine fault tolerance," presented at the 7th Symposium on Operating Systems Design and Implementation, 2006.

[16] I. Elgedawy, Z. Tari, and J. A. Thom, "Correctness-aware high-level functional matching approaches for semantic web services," *ACM Transactions on Web, Special Issue on SOC*, vol. 2, no. 2, May 2008.

[17] I. Elgedawy, "On-demand conversation customization for services in large smart environments," *IBM Journal of Research and Development*, Special issue on Smart Cities, vol. 55, no. 1/2, 2011.

[18] L. Lamport, "Fast Paxos," *Distributed Computing*, vol. 19, no. 2, pp. 79-103, 2006.

[19] Y. Mao, F. P. Junqueira, and K. Marzullo, "Mencius: Building efficient replicated state machines for wans," in *Proc. the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI'08*, 2008, pp. 369-384.

[20] Y. Amir, B. Coan, J. Kirsch, and J. Lane, "Customizable fault tolerance for wide-area replication," presented at the 26th IEEE Symposium on Reliable Distributed Systems, 2007.


**Islam Elgedawy** is an associate professor at the Computer Engineering Department, Middle East Technical University, Northern Cyprus Campus. He received his B.Sc. and M.Sc. degrees in computer science from Alexandria University, Egypt in 1996, and 2000, respectively, and his Ph.D. degree in computer science from RMIT University, Australia in 2007. His work focuses on the areas of service-oriented computing, organic computing, software engineering and distributed computing. He is an author and co-author of many publications in international journals and conferences, also he has a growing record of consultancy and professional services.