Rain4Service: An approach towards decentralized web service composition

Tanveer Ahmed, Abhinav Tripathi, Abhishek Srivastava
Discipline of Computing Science
Indian Institute of Technology Indore, India
Email: {phd12120101, ee130002001, asrivastava}@iiti.ac.in

***Abstract*—The widespread adoption of service oriented architecture owes its popularity to service composition, where several web services are combined dynamically at runtime. As is evident today, the Internet is evolving towards the '*Future Internet*'. In this context, web service composition has to deal with the problems of mobility, fault tolerance, reliability and the ultra large scale of the Future Internet. The practice of composition, the most popular variant of which is service orchestration, is expected to face numerous challenges in the Impending Future Internet. This is mainly because it comprises a central control point. Service choreography is widely viewed as an appropriate remedy to these problems as it has a predominantly decentralized orientation. In this paper, we propose a decentralized framework for the purpose of executing a case oriented workflow that would be ideal for the Future Internet. The services in the framework communicate and co-ordinate amongst each other without involving a centralized orchestrator. Further, we propose a technique that models the behavior of rain drops to achieve decentralized service composition. Based on the principles of message based service choreography, the proposed composition technique aids selection and execution of web services. We show how the model achieves service composition leveraging both static and runtime properties of a service. A runtime SOA test-bed to validate the decentralized framework and the composition technique is developed in JAVA. Validation is done via real web services. Multiple workflows are executed to demonstrate the viability of the model in actual deployment. Through experiments and exploration, the technique is found to outperform similar techniques in literature.**

I. Introduction

Service oriented computing has become a *pervasive* paradigm, where efficiency, in either a business process or a data intensive scientific application, is the center of attention. This computing paradigm owes its popularity to dynamic service centric systems, and their capability to combine multiple loosely coupled and platform independent services at runtime, thereby presenting an illusion of having a proficient and a dedicated application, provisioned for execution anytime and anywhere. The most favoured artifacts, for an Internet based scenario are web services, and their temporal collaboration, commonly referred to as web service composition. Web service compositions are a classic utilization of component based software engineering, where a complex application is built from loosely coupled, autonomous and platform independent interfaces. However, this paradigm mostly rely on a centralized orchestrator, e.g. VMware vCenter Orchestrator, Taverna (scientific workflows) etc., thereby bringing in a lot of potential issues (scalability, reliability, single point of failure, to name a few), that make the entire composition engine a heavy choke point. A single point of failure could render an entire hierarchy of services moot.

As outlined previously, the internet is evolving towards the '*Future Internet*'. One of the constituents of the Future Internet, Internet of Services (IoS) outlines, services can collaborate among each other to support a process spanning across different organizations [9]. In the context of the Future Internet, IoS has to deal with the issues regarding mobility, scalability, reliability etc., therefore reliance on a centralized architecture for either of the two scenarios (business or scientific) is a *'slippery slope'*. In this context, we believe service choreography could be '*the*' solution. However, enacting a business or a scientific process via service choreography is a challenge. The reason for which is outlined below.

Consider, a workflow is enacted via service choreography. In that case, the higher order abstract functionality is specified via a choreography description language, e.g. Let's Dance. The first requirement that arises, is to make each service understand and execute its respective role. Moreover, if a single service is interacting with multiple services, then it must preserve the contextual information for all the choreographies. Further, it has to become self-aware while instantiating the appropriate *dance* for a particular workflow. To accomplish this functionality, the services have to parse the description and store the context information. The current implementation of *autonomous* web services does not allow for such a requirement.

Second, consider services following the *RESTful* standard. Moreover, consider services hosted on a mobile device. In our view, the ideal choice is to access and host services using RESTful standards, thereby removing the heavyweight protocol of WSDL-SOAP (due to battery constraints). However, the present RESTful standard was founded on the pillars of statelessness. In this context, how do a *'stateless'* service access and pass data/parameters to a separate *resource* without storing *statefull* information? Composition of RESTful services in a decentralized environment is still a big problem.

Third, if services are communicating amongst each other, then should they pass data directly or share data via a shared memory? Is the shared memory reliable, and will it produce timely results (for time critical applications), with the amount

of data (42.07 exabytes/month in 2014[1]) generated and transfered over the existing network infrastructure. Further, in the Future Internet with cloud computing, or rather *federation of clouds*, allowing XaaS, how do different flavors of services (Human, RESTful, WSDL-SOAP) communicate and interact?

Some works in literature [12], [10] propose wrapping a service with an interpreter to achieve service choreography. A protocol to execute, enact, and verify choreographies in peer-peer network is proposed in [14]. However, the author states "*an extra layer of functionality, a choreography interface needs to be added to the stack*". The question that arises is: Do the existing service implementation allow for such a proposal? Even if a service is wrapped with an interpreter, the next question that arises is in the context of mobile services, with severe battery constraints. Addition of an extra middle layer on the device itself, will cause unnecessary power consumption. Moreover, the techniques do not talk about the scale, the economic factors, and the heterogeneity of the Future Internet. Further, the issue of reliability is not addressed. With mobile services, this issue is inevitable.

In this paper, we propose a decentralized deployment framework keeping in mind the issues highlighted above. The framework is capable of executing and selecting services at runtime. The services collaborate independently and communicate directly without involving a centralized orchestrator. Moreover, we propose a greedy and a lightweight composition method customized from the behavior of Rain Drops. The technique creates an environment that facilitates the selection of a service from a set of similar services. We inspect and discuss the behavior of real world rain, the effect of gravity, the roughness of a surface, the extent of corrosion, the trail left by a drop etc. to devise a system that produces the most optimal result to compose services together. We present our findings and utilize these traits to develop a composition model for web services. While formulating this technique, we assumed the surface on which a drop of water flows down is slanted. It is also assumed, the drop is falling and rolling (down a surface) under the force of gravity.

In this paper, experiments are conducted with real web services. Similar techniques in literature, [5] [6], do not include any prototype implementation. Furthermore, none of the issues highlighted above are addressed. Moreover, the proposed composition technique is found to outperform the two techniques.

The rest of the Paper is organized as follows: Section II introduces the decentralized framework and the composition technique. Results are presented in Section II. Related work is discussed in IV. Finally, we conclude in Section V.

For the purpose of clarification, Web services are referred to as service nodes or nodes throughout the paper. A set of services instantiated for a '*workitem*' or a *'process-step'*is called a level.

[1]http://www.cisco.com/en/US/solutions/collateral/-ns341/ns525/ns537/ns705/ns827/white paper c11-481360.pdf

II. Proposed Model

A. Workflow Description

To execute a workflow in a decentralized manner, we rely on execution rules. Using execution rules, and the fundamentals of domain specific language, any type of workflow can be mapped from a higher abstract level to a lower level with executable code. Or, techniques in literature, e.g. MAP [14], LetsDance can be used. A technique to devise implementations from a higher order choreography language is proposed in [16]. In this paper, we are discussing the model in the context of decentralization in the Future Internet, therefore there are some other requirements that must first be addressed. In the next subsection, we discuss some of the technological dependencies of the proposed work.

B. Technology and the Future Internet

In the proposed work, we rely on some of the existing technology available in the open-source community and the existing work in literature [11], [2], [13], and Distributed Shared Memory. We lay the technological foundation of the model on these works.

In the Future Internet, there is an inevitable need of a modern service description language. The service description language should be powerful to describe both software services and human based services. Moroever, the language must be self-sufficient to cater to the needs of the IoS. In the IoS, services are tradeable and marketable. Moreover, an individual service will become capable of executing a workflow on its own. The ideal candidate realizing all the requried functionalities, is the Unified Service Description Language (USDL) [11]. In our view, the language is comprehensive enough to describe and cater to the needs of the IoS. USDL is a protocol suite describing multiple modules viz Pricing, SLA, Legal, Functional etc. The services described via USDL are bundled as 'Network Provisioned Entities'. The service can either be a stand-alone service or a 'bundle' of multiple services. USDL allows the customers, who are the ultimate stakeholders, to rate a service or a *service bundle*. Moreover, USDL is capable of describing both human provided services and software based services (RESTful and SOAP). In IoS, the services are the economic entity, thereby involving international legal aspects. Since all of the major requirements are satisfied by USDL, therefore in the proposed work, we rely on this language for service description.

To cater to the requirements of a stable delivery platform (for service access), a Distributed Federated Enterprise Service Bus (hereafter, DSB or bus) is employed. A prototype DSB is proposed in [2]. However, to deal with the issues of service choreography (outlined previously), the present implementation is not sufficient. In this paper, we extend the present prototype to introduce certain modifications. First, the Binding Component (BC), as proposed in [2], is modified to interpret the rules (or a choreography description) to execute a workflow. Second, the listener component (of the DSB) for each web service is modified to translate messages

between various flavors of services (SOAP, RESTful, Human Provided Services), a similar method is available in [13]. The '*default*' listener component is attached by the DSB for every SOA '*app*' deployed on the bus. In the proposed work, the component also acts as a container, storing the essential context information. We rely on the DSB's multi-node architecture to address the scalability issue of the Future Internet. Because of the existing technology and the proposed changes, the DSB could be the best Middleware (at present) to execute Fututre Internet's SOA apps, without introducing an extra layer on the services.

As far as the communication overhead is concerned, the services interact via an event driven publish-subscribe model [1]. They subscribe to listen to the Force value (described later), a node is offering. Further, if there is any change in the execution procedure, only the concerned resources are notified. Hence, an increased level of privacy. Since, the event-driven mechanism of an ESB (or DSB) is well appreciated in both academia and industry, the model relies on these features to conduct transmission (of data) related affairs. It should be pointed out that to cater to the requirements of the proposed model, some triggering rules (in JavaScript or 4GL) have to be hooked in the event bus[2], thereby making the strategy feasible in actual deployment.

The next issue is reliability, especially with mobile web services. It was outlined in the previous paragraph, the model relies on the Middleware. It is a known fact that a DSB or an ESB, offers intelligent content based routing and guaranteed delivery of messages. However, for realtime applications, the delivery mechanism will cause a lot of unnecessary delay, especially when the deployed application is temporarily unavailable (The DSB offer two 'message transporter' properties: Fast and Reliable[3]). In the reliable mode, the DSB waits for the unavailable app to be hooked in again. A very simple solution is to keep redundant services available for *immediate* execution. These services also allow a capability for parallel processing, load balancing, and dynamic adaptations.

To provide a stable storage location, we rely on the novel concept of the Semantic web data space, e.g. OpenLink Data Spaces (ODS)[4] in the open source community. It is a next generation application of distributed collaboration, satisfying the needs of distributed shared state, client-side updates and data consistency[5]. It also allows for an inexpensive method to create *Linked Data Web*. The ODS type storage schema allows a simple API to read and write to a persistent storage. The APIs to access the protocol are exposed as a web service (SOAP & RESTful APIs). The data center allows the query to specified in GData, OpenSearch, XQuery/XPath over HTTP, SPARQL etc. These type of storage spaces bring in the SOA style of communication and storage patterns, hence they are ideal for the Future Internet apps. A question could arise here: Why use a semantic space instead of just a simple Distributed Shared Memory? A space like ODS offer a semantically rich, a stable SOA style distributed semantic repository, deals with the virtualization of heterogeneous data-centers accessible via existing protocols, is deployable on a cloud based infrastructure. So, the real question is: Why not?

In our view, using the proposed technological improvements and modifications, service choreography can be enacted without requiring significant changes in the existing infrastructure. The proposed methodology is validated with a prototype using real web services. It should be noted here that in this paper, our motivation is to propose a technological solution to achieve a decentralized service composition only. Verifying and recovering from a choreography makes room for future development.

C. Rain4Service: Decentralizing Web Service Composition

Until now, we have laid the technological foundations of the decentralized execution framework. In this section, we discuss the rain model for service composition. We start by discussing some of the observations made during analysis.

In the real world, whenever the individual droplets of rain fall, they have a tendency to come together and form bigger drops. Whenever a drop of water touches the surface, it starts rolling immediately (under the influence of gravity). While the drop is in motion, it exhibits certain interesting characteristics:

i) Assuming a drop to be spherical, initially the radius is large. On its way towards a sink, the radius of the drop decreases continuously. This is due to the fact that the drop leaves behind a trail of liquid as it rolls down.

ii) If the surface offers high friction, a situation may arise when the drop stops moving completely and forms a bulge on the surface.

iii) A drop always rolls down a surface leaving behind a trail of liquid. The subsequent drop, that follows the same trail moves more rapidly. The reason is quite simple: the layer of the liquid left behind, reduces the frictional drag between the new drop and the upper layer of the material.

iv) It is a common observation, that when rain occurs drops fall and roll under the influence of gravity. It was also observed, there existed certain situations when bulges containing liquid droplets were formed. We observed, there were instances when drops kept coming into these bulged areas, still the bulge showed no mobility at all i.e. no droplet flowed out.

Looking at the inherent structure of a surface, there are a lot of inferences that can be drawn. It is observed that sometimes a rough patch is formed due to wear & tear. The roughness of a surface can be explained in terms of the availability of cracks. A rough patch is made up of multiple lines of cracks '(LoCr)'. The more the LoCr, the rougher and broader a patch. Next, if we consider a surface to be divided into various zones (as in a voronoi graph), then a drop of water has to travel through various zones to finally end up in the sink.

[2]http://www.oracle.com/technetwork/articles/soa/jellema-esb-090659.html

[3]https://doc.petalslink.com/display/petalsesb31/Petals+Enterprise+Service+Bus

[4]http://ods.openlinksw.com/wiki/ODS/

[5]http://www.cs.duke.edu/ari/cisi/relay/html/paper/paper.html

While it is raining, the drops have a natural propensity to cause stagnation on the surface. The objective of a *road-planner* is to make the drops end in the sink quickly to avoid such scenarios. Though a lot of observations are not discussed, we describe only the ones that are relevant to the model.

D. The Frictional Drag

In the proposed model, each service node offers a frictional drag. A drop of water is considered as the composite application request. In order to avoid stagnation, the current drop(s) must leave the zone(s) quickly. In the proposed work, it is analagous to a composite application request leaving a service node immediately. In other words, the waiting time experienced at a service must be minimum. This solves the dual purpose of congestion avoidance and fast application completion. Further, we believe the Future Internet will be capable of processing *Big Data* via SOA, deployed on a cloud based infrastructure [15], [3]. We believe, the passage of big data among different resources (or services) will also become an important QoS parameter in the Future Internet. Therefore, we propose the metric *flow time*, defined as "*the time required to pass data and control from one resource to another resource*". Flow time will depend on the conditions of latency, bandwidth availability, geographical distribution, reliability, uptime etc. Further, it is a known fact that the processing of Big Data requires substantial time, therefore the parameter, *waiting time* is also important. Based on the two parameters, the definition of the frictional drag due to the waiting time and the flow time is as follows:

$$f_w = t_w j + t_f(ij) \tag{1}$$

where, f_w is the frictional drag offered by a node (e.g. S_j). $t_w j$ is the waiting time experienced by a request, $t_f(ij)$ is the flow time between the current node (i) and successor node(j).

A special case arises here, when the same resource realizes two different and subsequent activities of a workflow. In such a case, the flow time will become negligible. Hence, a reduction in the frictional drag is achievable.

In the previous subsection, we discussed the role LoCr plays in determining the roughness of a surface. It is a known fact that if a surface is rough, then it offers a high frictional drag. In the rain model, the frictional drag offered by a service is also expressed in terms of the functionality a USDL based service offers. The motivation to include this parameter is explained below.

Consider a situation where a service (S) exposes multiple *'functions'* (USDL's terminology) via its USDL interface, e.g. PowerSearchRequest, AuthorSearchRequest etc. If a customer wants to search for an author of a book, then the best choice is to invoke the AuthorSearchRequest function. The results obtained will be in the context of 'Books' and 'Authors'. If one compares this to a service which offers only a basic 'search' function, then result obtained would be difficult to analyze. Moreover, it will require further data manipulation. In this case study, the service S, has provided granular support to its customers, thereby making the execution of a problem easier.

Second, if a service (S1) has, e.g. 10 functions and another service (S2) has 2 functions, then S1 is capable of catering to 10 users simultaneously. In today's world, IT organizations strive to achieve efficient resource utilization. In this context, multiple functions offered by a service can help achieve a high concurrency and a high degree of parallelism with optimal usage of the underlying hardware. Third, our inspiration comes from one of the basic principles of software engineering - function point analysis (FPA). "The capabilities of FPA allow accurate estimates to be made, risks to be evaluated, and project scope to be negotiated, before final commitments are made"[6]. It is understood that a web service is software and its functionality can be measured in terms of the functions it has to offer.

To incorporate the frictional drag due to functions specified, we have assumed that there are a fixed number of cracks in each zone on the surface. The number of functions defined by a service's Interface act as filler to fix the cracks. Mathematically,

$$LoCr = \theta - \mu \tag{2}$$

where, θ is the fixed number of cracks assumed, μ is the number of functions offered by a service. The friction offered due to the number of functions is defined as:

$$fo = LoCr \tag{3}$$

It should be pointed out that this frictional drag might not be of any importance to a normal human being. However, this drag is of special interest to intermediaries, resource aggregating agents, IT or business organizations requiring either gray or black box views, to enrich existing infrastructure, enhance functionality etc (USDL, the functional module).

To combine the two frictional drags, a computationally inexpensive linear combination strategy was employed. The combined frictional drag offered by a node is defined as follows:

$$f = \alpha \frac{fw}{T} + (1 - \alpha) fo \tag{4}$$

where, α is the bias parameter in the range [0,1], T is a constant, added in the equation to make the two frictional drags addable (Since, consistent dimensions are required for addition, *fo* is dimensionless).

It is a known fact that all the drops of water have the same density. Now, considering all the drops have same size, we can say their mass is same. Therefore, the force of gravity *mg*, is the same for all the drops. The resultant force (or just force) experienced by a drop (or a request) while drifting is equal to

$$F_r = F_g - f \tag{5}$$

where, F_r represents the resultant force, F_g is the force of gravity and f is the frictional drag offered by a service

[6]http://www.totalmetrics.com/function-point-resources/whatarefunctionpoints

node. It should be noted, the value of F_g is common for all the nodes and requests. Using the concepts of *motion* in classical physics, the request will move to a service node that has the maximum resultant force. Therefore, each subsequent request is passed to a lower level node so that the resultant force is maximum.

$$F_x > \forall y F_y; x, y \in level(i); x \neq y \tag{6}$$

where, F_x *and* $\forall y$ F_y are the resultant forces offered by Services S_x *and* S_y(Service set) at the same level *i*. *level*(*i*) is the set of all the nodes offering similar functionality.

As outlined earlier, we have used event based updates. In the model, it could happen that a service does not send its force value at all. It implies the service is dead and there are reliability problems. This situation is similar to the immobile bulge formation on the surface i.e no matter how many requests come in at this node, none shall move ahead. We also discussed the importance of the trail left behind by a drop to reduce the frictional drag. In the model, it is analogous to the fact that the updated force value a node is offering has not been received yet. Hence, there is no additional computation involved (extra processing time for updating force value in the force table) to send out the next request. In the results section, the importance of event based updates, update interval, the processing overhead etc is discussed.

E. Observations in the Proposed Model

1) Observation I: We present a mapping of force value as a function of event time. By event time, we imply an occurrence of a business rule, a system event, a bus event etc. Consider a node S_i at the upper functional level (the workitem set) *n*, wants to send a request to the chosen node S_j, at the lower functional level *n+1* (the next workitem set). Further, consider at time *t* the queue size of S_i is σ and queue size of S_j is τ. Now, at time t+δ, ϕ and β requests arrived and completed at node S_j . According to equations (4) and (5), the force value for the two cases is shown below.

$$F_rj(t) = F_g - \alpha\frac{(\tau t_s(j) + t_f(ij))}{T} + (1-\alpha)(\theta - \mu) \tag{7}$$

or,

$$F_rj(t+\delta) = F_g - \alpha\frac{((\tau + \phi - \beta)t_s(j) + t_f(ij)}{T} + (1-\alpha)(\theta - \mu) \tag{8}$$

Subtracting, 7 from 8, we get

$$F_rj(t+\delta) - F_rj(t) = -\alpha\frac{(\phi - \beta)t_s(j)}{T} \tag{9}$$

or,

$$F_rj(t+\delta) = F_rj(t) - \alpha\frac{(\phi - \beta)t_s(j)}{T} \tag{10}$$

This implies, if the rate of arrival of new requests is more that the request completion rate, then the force value will decrease. Otherwise it will increase. The rate of increment or decrement is governed by the parameter α. Therefore, the parameter α plays an important role in service composition. The importance of α is also discussed in the results section.

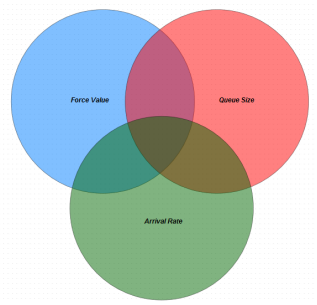

Figure 1: InterRelation Diagram

Based on the discussion in this subsection, it can be deduced that the three entities viz. the force value, the queue size, and the arrival rate are interrelated.

III. Results

For comparing the composition technique, we have chosen three other techniques viz. [5], [6] and Queue size Based Selection. The work in [5] and [6] are the two closest techniques in literature utilizing the waiting time criteria for service composition. A runtime SOA testbed was developed in JAVA language for the purpose of experimentation. The services exchanged simple strings and performed some processing. While configuring the experimental setup, it was observed that the execution time of each service and the flow time between each pair of service is less. Therefore, to simulate real world behavior, the invoking thread of each service was made to sleep for a random amount of time. The experiments were conducted on a laptop computer with Intel i7 Quad-Core processor, 2.4 Ghz and 8 GB RAM. It is worth pointing out, the testbed with a DSB and a true Semantic data space is left as a part of the future work. Therefore, stubs were written for both the BC and the listener components outlined above. However, a DSM (MozartSpace[7]) (distributed over multiple nodes) deployed as an SOA application was used as a stable storage location. For the purpose of comparison, Figure 2A was executed 1000 times. Though, the Figure depicts a simple workflow with redundant services, our motive was to check the feasibility of the proposed solution in decentralized execution and equitable load distribution. The application container for the web services was Apache Tomcat v7.0.41.

A. Behaviour of the Completion and Waiting Times

In Figure 4, the Standard Deviation of the proposed model w.r.t other techniques is presented. As visible from the Figure, the standard deviation for the proposed model, compared to other three techniques, is less. Hence, the proposed model achieved equitable load distribution without compromising the quality of experience of a user. Leitner *et al* states "The user needs quality metrics which describe the quality of the business transactions in an end-to-end fashion" [7]. *In our view, the waiting time and the completion time are one of the metrics*. Moreover, the stability of a

[7]http://mozartspaces.org

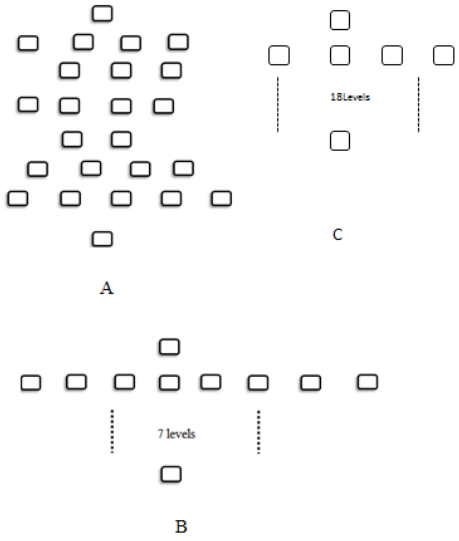

Figure 2: Experimental Workflows

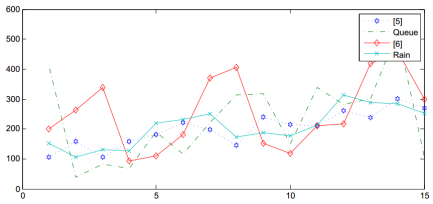

Figure 3: Request Completion Time

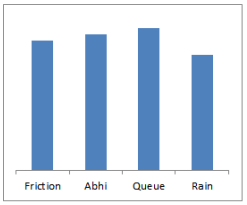

Figure 5: Average Completion Time

issues, only average completion time is presented. As visible, a low value of average completion time was observed when α is equal to 0.2. However, the optimal value of α kept changing with different configurations. But, it was noticed that the optimal value of α was always between [0.2, 0.6]. Theoretically, we expected a low completion time when α was 1. However, owing to the parameter of the flow time, an optimal value was observed only after exploration and experimentation.

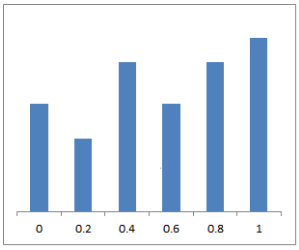

Figure 6: Average Completion Time Different α

system is thoroughly tested in times of severe load. The fact that the proposed model balanced load efficiently implies no node is overloaded, hence one can achieve and maintain the stability without compromising execution efficiency, which is crucial in todays IT environments.

Next, in Figure 3 the completion time for different techniques is presented. The figure demonstrates only a few events (for the purpose of visual clarification). It is observed, there exists a few situations where the completion time of the proposed model is high. However, in Figure 5 the average completion of all the events is shown. It can be seen from this Figure that the proposed model produces a low value of average completion time. Hence, the model also achieved fast job turn-around time. It should be noted here, that the two techniques [5] and Queue based selection model, does not take the criteria of flow time into consideration. This assumption is rather unrealistic. Since, in the future internet we believe due to network constraints passage of data between two resources will be difficult. Hence, considering this criteria while selecting a service will help in a quick turnaround time.

B. Impact of α

During analysis, several different configurations of α were tested. Different values of α, with a step size of 0.2 were chosen for the purpose of experimentation. In the experiments conducted, standard deviation, completion time etc was also calculated. Due to space constraints and visual clarification

C. Impact of Different Event Times

In addition to the experimentation with various combinations of α, experiments were conducted with different values of event time. In the proposed model, event time is important because it is at this time the force values are exchanged. A variation in the '*Force Table*' at the local resource will cause dynamic reconfigurations of the composition structure. Hence, experiments were conducted to test the importance of the event time as well. In Figure 7, we have shown average completion times with various values of event time. We expected the average completion to be minimum when the event time is less. But, in the results a low completion time was achieved when the event time was 4 seconds. It was found that beyond the limit of 8 seconds the values kept increasing. From this observation, it can be conclusively said that the update interval must neither be short nor long.

D. Communication Overhead

We also calculated the communication overhead involved during execution in terms of data exchanged between services for the Figures 2B and 2C . The amount of data exchanged is compared with a centralized orchestrator controlling and selecting services. Different sizes of data was chosen for the purpose of experimentation viz. 30 bytes, 90 bytes and 361 bytes. The data analysis was via the tool Wireshark[8] with Ubuntu 12.04 LTS OS.

[8]http://www.wireshark.org

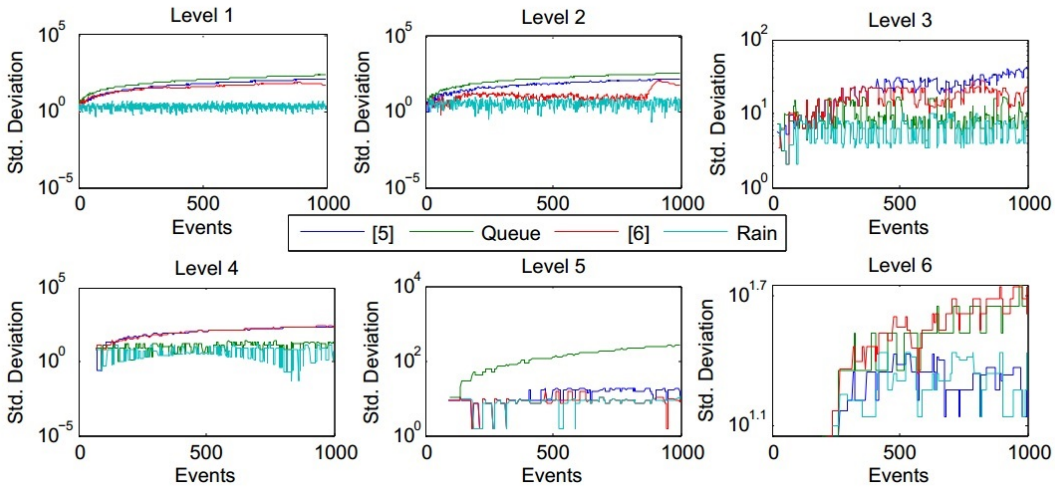

Figure 4: Standard Deviation Waiting Time

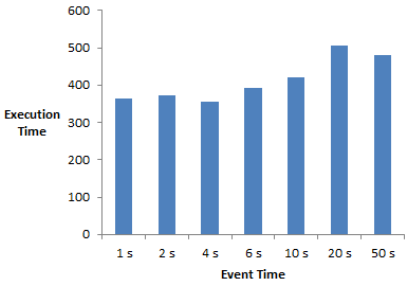

Figure 7: Average Completion Time with Different Event Time

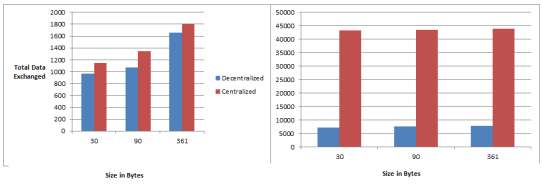

Figure 8: Data Exchange Service Domain B and C

The total amount of data exchanged is shown in Figure 8. As visible, the amount of data exchanged is less in the decentralized scheme. Therefore, the model not only executed the application quickly, but also requires a less amount of data exchange. It was also observed that amount of data exchanged kept increasing for a centralized orchestrator when the size of a workflow increased. From this observation, we can conclude that when exchanging 'Big Data' via a centralized scheme, not only the execution time will increase, but the load on the network will increase as well. In the context of the Future Internet, this is a troublesome issue. Hence, to execute scientific applications a decentralized scheme is far better than the centralized scheme.

IV. Related Work

In this section, we discuss some of the related work in the field of service composition. Very close to our work are the two techniques [5] and [6]. These techniques also focus on the parameter of the waiting time. [5] is a technique based on queuing theory where concepts are customized to cater to the requirements of service composition. The technique enabled a rough estimation of the waiting time expected at a Web service. Similar to [5], is a model presented in [6]. Here, a technique borrowed and customized from real world friction is presented. The technique also accommodated the parameter of waiting time. However, it was shown in the results section that the proposed model, outperforms both the two predecessors. A dynamic QoS aware subjective-objective approach for 'optimal strategy configuration determination' is proposed in [17]. The approach showed good performance over existing traditional strategies. A self-* framework for configuring and adapting services at runtime was proposed in [18]. The framework, PAWS, delivered self-optimization and guarantee guaranteed service provisioning even in failures. In literature, various techniques also focus on the metric of end-to-end delay, with network latency to compose services in a cloud based environment [19], [20]. We also focus on such parameters via the proposed metric 'flow time'.

Nature inspired metaphors have recently caught some attention in the services sector. Inspired from such metaphors a technique is presented in [12]. Similar to our rain metaphor, the author in the paper focus achieving a decentralized workflow execution with a chemistry metaphor. However, they do not focus on balancing load among similar services. Very close to the work presented here is [10], the authors focus achieving orchestration and choreography with the chemistry metaphor. However, they relied on conventional workflow execution mechanism. We on the other hand, proposed a new framework for decentralized workflow execution. We also relied on a decentralized DSM for parameter exchange.

Further, we proposed a novel composition mechanism. No dynamic selection was done in [10], moreover a single service was instantiated for a workitem. A technique to enact, verify, and execute a service choreography is presented in [14]. However, similar to [12] and [10], the author introduced an extra choreography layer, requiring major changes in the underlying implementation of web services. We on the hand, achieved choreography with the existing infrastructure. Moreover, we focused on the composition from a greedy point of view. The technique do not talk about load balancing and dynamic adaptations. Also, the choice of services executing the workitems is fixed.

V. Conclusion and future work

In this paper, a decentralized deployment framework and a composition technique was proposed. The experiments were conducted via a SOA test-bed. Preliminary testing demonstrate good performance of the proposed solution. The model in based on the state-of-the-art Middleware and Semantic space to achieve service choreography. The composition technique outperformed similar techniques in literature in terms of average waiting time and job completion time. Future work includes a full scale development of the prototype with a true semantic space and DSB. Further, the execution of the web services and their composition will be accomplished on a grid based facility, thereby studying the effect of geographically distributed nodes, real world low bandwidth and high latency environments.

References

[1] Marechaux, Jean-Louis. "Combining service-oriented architecture and event-driven architecture using an enterprise service bus." IBM Developer Works (2006): 1269-1275.

[2] Baude, Franoise, Imen Filali, Fabrice Huet, Virginie Legrand, Elton Mathias, Philippe Merle, Cristian Ruz et al. "ESB federation for large-scale SOA." In Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 2459-2466. ACM, 2010.

[3] Zimmermann, Alfred, Michael Pretz, Gertrud Zimmermann, Donald G. Firesmith, and Ilia Petrov. "Towards Service-Oriented Enterprise Architectures for Big Data Applications in the Cloud." In Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013 17th IEEE International, pp. 130-135. IEEE, 2013.

[4] Issarny, Valrie, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadist, Marco Autili, Marco Aurlio Gerosa, and Amira Ben Hamida. "Service-oriented middleware for the Future Internet: state of the art and research directions." Journal of Internet Services and Applications 2, no. 1 (2011): 23-45.

[5] Srivastava Abhishek, and Paul G. Sorenson. "Utilizing the Waiting-time Criterion for Selecting Services in a Composition Scenario." In Services Computing (SCC), 2010 IEEE International Conference on, pp. 258-264. IEEE, 2010.

[6] Ahmed Tanveer, and Abhishek Srivastava. "Minimizing Waiting Time for Service Composition: A Frictional Approach." In IEEE 20th International Conference on Web Services (ICWS), 2013, pp. 268-275. IEEE, 2013.

[7] Leitner, Philipp, Anton Michlmayr, Florian Rosenberg, and Schahram Dustdar. "Selecting web services based on past user experiences." In Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific, pp. 205-212. IEEE, 2009.

[8] Leite, Leonardo AF, Gustavo Ansaldi Oliva, Guilherme M. Nogueira, Marco Aurlio Gerosa, Fabio Kon, and Dejan S. Milojicic. "A systematic literature review of service choreography adaptation." Service Oriented Computing and Applications (2012): 1-18.

[9] Cardoso, Jorge, Konrad Voigt, and Matthias Winkler. "Service engineering for the internet of services." In Enterprise Information Systems, pp. 15-27. Springer Berlin Heidelberg, 2009.

[10] Wang, Chen, and J. Pazat. "A Chemistry-Inspired Middleware for Self-Adaptive Service Orchestration and Choreography." In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, pp. 426-433. IEEE, 2013.

[11] Cardoso, Jorge, Alistair Barros, Norman May, and Uwe Kylau. "Towards a unified service description language for the internet of services: Requirements and first developments." In Services Computing (SCC), 2010 IEEE International Conference on, pp. 602-609. IEEE, 2010.

[12] Fernandez, Hctor, Cdric Tedeschi, and Thierry Priol. "A Chemistry Inspired Workflow Management System for Decentralizing Workflow Execution", IEEE Transactions on Services Computing, doi: 10.1109/TSC.2013.27 (pre-print).

[13] Leitner, Philipp, Florian Rosenberg, and Schahram Dustdar. "Daios: Efficient dynamic web service invocation." Internet Computing, IEEE 13, no. 3 (2009): 72-80.

[14] Barker, Adam, Christopher D. Walton, and David Robertson. "Choreographing web services." Services Computing, IEEE Transactions on 2, no. 2 (2009): 152-166.

[15] Koulouzis, Spiros, Reginald Cushing, Konstantinos A. Karasavvas, Adam Belloum, and Marian Bubak. "Enabling web services to consume and produce large datasets." Internet Computing, IEEE 16, no. 1 (2012): 52-60.

[16] Su, Jianwen, Tevfik Bultan, Xiang Fu, and Xiangpeng Zhao. "Towards a theory of web service choreographies." In Web Services and Formal Methods, pp. 1-16. Springer Berlin Heidelberg, 2008.

[17] Zheng, Zibin, and Michael R. Lyu. "An adaptive QoS-aware fault tolerance strategy for web services." Empirical Software Engineering 15, no. 4 (2010): 323-345.

[18] Ardagna, Danilo, Marco Comuzzi, Enrico Mussi, Barbara Pernici, and Pierluigi Plebani. "Paws: A framework for executing adaptive web-service processes." IEEE software 24, no. 6 (2007): 39-46.

[19] Ye, Zhen, Xiaofang Zhou, and Athman Bouguettaya. "Genetic algorithm based QoS-aware service compositions in cloud computing." In Database Systems for Advanced Applications, pp. 321-334. Springer Berlin Heidelberg, 2011.

[20] Klein, Adrian, Fuyuki Ishikawa, and Shinichi Honiden. "Towards network-aware service composition in the cloud." In Proceedings of the 21st international conference on World Wide Web, pp. 959-968. ACM, 2012.