Membrane Computing Inspired Approach for Executing Scientific Workflow in the Cloud

Tanveer Ahmed, Rohit Verma, Miroojin Bakshi, Abhishek Srivastava

Indian Instititue of Technology Indore
{phd12120101, phd12110101, cse1200105, asrivastava}@iiti.ac.in

Abstract. The continuous expansion and appreciation of the service oriented architecture is due to the standards of loose-coupling and platform independence. Service-Oriented Architecture is most commonly and effectively realized through web services, and their temporal collaboration commonly referred to as web service composition. In the present scenario, the most popular variant of composition is service orchestration. Orchestration is achieved through a centralized '*heavyweight*' engine, the orchestrating agent, that makes the deployment configuration a massive '*choke-point*'. The issue achieves significance when data and compute intensive scientific applications rely on such a centralized scheme. Lately, a lot of research efforts are put in to deploy a scientific application on the cloud, thereby provisioning resources *elastically* at runtime. However, just like the orchestrating agent it is the cloud now that provisions resources and in a similar manner there is an unnecessary burden on the hosted platform. In this paper, we aim at eliminating this central 'choke' point by presenting a model inspired from '*Membrane Computing*' that executes a scientific workflow in a decentralized manner. The benefit of this paradigm comes from the natural process of autonomy, where each cell provision resources and execute process-steps on its own. The approach is devised keeping in mind, the feasibility of deployment on a cloud based infrastructure. To validate the model, a prototype is developed and real scientific workflows are executed in-house (within the Intranet). Moreover, the entire prototype is also deployed on a virtualized platform with software defined networking, thereby studying the effects of a low bandwidth environment, and dynamic provisioning of resources.

Keywords: Scientific Workflows, Service Oriented Architecture, Membrane Computing

1 Introduction

Service oriented architecture has now become a multipurpose paradigm executing business processes and helping compute intensive scientific applications [2]. It's success can largely be attributed to advancements in the field of cloud computing. The technological advancements, notwithstanding, the paradigm relies

on a centralized solution to orchestrate either a business process or data intensive scientific applications as the case may be. The orchestrator, by virtue of its inherent centralized nature is a single point of failure and suffers from the issues of scalability, reliability, fault tolerance, security, privacy [4].

It is well known that a scientific workflow is resource intensive, whereas a business workflow is control oriented. A scientific workflow is well exemplified by the LSST[1] experiment aimed to recursively explore and take images of the sky for a period of 10 years and expected to generate data (300MB/s) in the order of exabytes (both raw and pruned). To rely on a centralized architecture in this context, is a potential processing and communication bottleneck. Moreover, the failure of an orchestrator or the hosted platform (cloud's IaaS), causes a cascaded chain reaction, making the entire hierarchy of deployed services moot. The issue is evident by the failure of Amazon in 2011, 2012, and 2013. A decentralized setup, on the other hand, achieves a reduction in failures and enables quick recovery. Considering the example of AstroGrid science (calculating Redshift) presented in [1], a reduction of 43.47% (data transfer) was achieved by enacting a workflow using a decentralized architecture. Hence, in the context of scientific applications, we believe a fault tolerant decentralized solution towards workflow execution should be the way forward.

With the advent of cloud computing, concepts such as *Elastic Computing*, *federation of clouds*, and *Mobile Cloud Computing*, are just around the corner. In respect to the scientific workflows, the benefit of decentralization comes in the context of Elastic Computing. Consider, a compute intensive workitem is processing huge volumes of data. If a centralized orchestrator is handling this task, then the orchestrator is responsible for gathering resources, provision them into the existing workflow, perform migrations at remote locations and several others resulting in an un-necessary burden on the controlling authority. Moreover, a centralized engine foster a lot of infrastructure, development, maintainence and operational costs. Therefore, for certain scientific experiments limited by budget constraints, the decentralized architecture is an ideal candidate.

Bell *et. al* [3] states "the rapidity with which any given discipline advances is likely to depend on how well the community acquires the necessary expertise in database, '*workflow management*', visualization, and cloud computing technologies". Taking this line of reasoning as a benchmark for the proposed work, we propose a decentralized solution to execute a scientific workflow via membrane computing approach. We take inspiration from biology to design a workflow management model that execute services, realizing the *process-steps* or *workitems*, in a decentralized manner. We use the elementary principles of membrane computing to model the execution of a workflow. The choice of this type of an architecture comes with the benefit that it provides a natural and an intuitive method to model workflows, pass parameters and elastically increase or decrease resources at runtime. We consider each membrane as a service capable of executing a scientific workitem. The membranes are self capable of discovering other resources (or services) on their own. Each service is provided with

[1] http://www.lsst.org/lsst/science

evolutionary rules (Membrane terminology), it is through these rules the services evolve, allocate & provision extra resources, and execute their respective tasks in a cost-efficient manner.

Substantial literature focuses on achieving scientific workflow execution over the cloud [1], [2], [9], [11], [12]. However, most of these rely on the cloud (or the orchestrator) to elastically increase or decrease resources at runtime, or don't focus on elasticity at all. Therefore, our objective is to introduce '*autonomy*' in resource provisioning by the work-items themselves.

To the best of our knowledge, this is the first endeavour to explore the possibility of executing a workflow via Membrane Computing. To demonstrate the viability of the same in actual deployment, we have developed a prototype with 'real' services. We execute real scientific workflows collected from myexperiment[2], and deploy the prototype on a virtualizated platform (XENServer) to test the validity of the proposed work. During validation, the services exchange data and parameters via a stable storage . The stable storage itself was offered as a service, thereby affirming to the standards of Service-Oriented Architecture. Moreover, using Software Defined Networking, we study the effects of limited bandwidth capabilities during execution.

The contribution of this paper is two-fold:

1) A novel membrane inspired approach for decentralizing workflow execution, with autonomous provisioning of computing resources

2) A proof of concept, via actual deployment and execution of scientific workflows.

The rest of the paper is organized as follows: Section II presents a brief introduction to membrane computing. Membrane Inspired Scientific Workflow execution is discussed in Section III. Results are discussed in Section IV, Related Work is presented in Section V, We conclude with Future Work in Section VI.

For the purpose of clarification, a scientific workflow is called a workflow throughout this paper. A resource is analogous to a service instantiated on a Virtual Machine.

2 Membrane Computing Paradigm

Before beginning the discussion of Membrane Inspired workflow management, we present a small discussion on the Membrane Computing paradigm.

Membrane computing takes its inspiration from a living cell. A living cell is encapsulated by a membrane that separate its internals from the external environment. The cell encloses multiple natural artifacts, e.g. nucleus, golgi apparatus, molecules, vesicles etc. The cytoplasm hold charged ions, whose movement (either inwards or outwards) is controlled by the presence of certain type of proteins. Using chemical receptors, the membrane allows a selective passage of molecules and achieves cell-to-cell signaling.

The pioneering work in the area of membrane computing was proposed in [6]. The author proposed, the basic structure of a membrane consist of several

[2] http://myexperiment.org

separate sub-membranes. The membranes consist of the delimiting region, called multiset, where several different objects are placed. The evolution, manipulation, and transformation of objects is accomplished via evolutionary rules. The objects are transferred from one membrane to another membrane, causing transitions and carrying out the intended tasks. The execution rules are chosen in a non-deterministic manner, thereby presenting an illusion of having infinite parallel computing capability. The application of these rules is conditional i.e. a rule is invoked if certain reaction conditions are applicable. The rules as explained in [6] are of the form $a \longrightarrow b$, where a and b represents multisets of objects. Since, the data and objects are transferred from one membrane to another, the author proposed the notion of 'target indications'. Using target indications, the objects are retained, transferred and consumed. It can be deduced that using these rules the multiset can be written very easily. An example of a rule applied towards object evolution is demonstrated below.

Consider a rule of the form *(ij)* $\longrightarrow$ *(i,here) (i,out) (j,in) (j,out)*. In this example, a copy of i and j is consumed and two copies of i and j are produced. One copy of i is retained within the same membrane (the 'here' indicator), while the other one moves out to the surrounding environment (the 'out' indicator). Out of the two copies of j produced, one goes to the surrounding environment and the other moves inwards toward the inner membrane(s). There exists catalytic rules demonstrating the applicability only in the presence of a certain type of an object, e.g. $cb \longrightarrow cv$, where c is the catalyst. Also, there are non co-operating rules, e.g. $a \longrightarrow b$, membrane dissolving rules, e.g. $j \longrightarrow o\delta$, where δ denotes the membrane dissolving action. It should noted here, the author [6] deliberately points out that the membrane dissolving rule cannot be applied to the skin membrane (for obvious reasons). Further, there are communication rules, symport and antiport, demonstrating how membranes communicate. As outlined earlier, in the real world the membranes communicate via protein channels. Therefore, the protein channel and the molecules are the agents of communication in membrane computing. The *'symport rules'* allows for the passage of molecules in one way. On the other hand, the *'antiport rules'* allow for a two way communication via molecules.

There also exists membrane division and merging rules. A membrane division rule is of the form $[_1 a]_1 \longrightarrow [_2 b]_2 [_3 c]_3$ (a membrane is denoted as '[ ]', [6]), while a membrane merging rule is of the form $[_2 e]_2 [_3 f]_3 \longrightarrow [_1 d]_1$. Further, there exists endocytosis rules, exocytosis rules, gemmation rules etc. The rules are applied locally in each membrane in a non-determinstic, maximally parallel manner, thereby causing *transitions* in the system.

In this paper, we try to use these elementary concepts to achieve a decentralized workflow execution. Based on the discussion so far, it is understood that this paradigm has a natural orientation, and can execute any type of computation problem (e.g. SAT [14], TSP [15] etc). It is due to this feature, it has received a lot of attention in literature, right from the moment of its inception. It can be deduced that the membrane computing paradigm allows a natural metaphor and an intuitive mechanism to model the complex behavior of scientific workflows.

As discussed previously the paradigm allows communication rules (symport and antiport), membrane dissolving rules, membrane division and merging rules etc. Using these rules, scientific workflow constructs and functionality can be managed with little efforts. Further, applying the evolutionary rules, a workflow itself can be modified dynamically (via endocytosis, exocytosis, gemmation etc.).

3 Membrane inspired scientific workflow execution

In the proposed work, a membrane is considered as a service capable of realizing a single 'workitem' or a '*process-step*'. Each membrane has its fluid (local memory) capable of storing the contextual information and local data (or molecules). The contextual information includes the load-indicator parameters, the inner membranes (the successor workitems), the outer membranes (the predecessor workitems), the resource pool etc (discussed below). The membranes communicate via the 'symport' rules to pass control to the subsequent membrane. The objects and data are passed via the multiset. The data is equivalent to the proteins capable of penetrating the membrane structure. The membranes do not pass data directly, but rather direct the subsequent membranes to read from the stable storage location (implemented as a scalable distributed shared memory). In the following text, a *service* is called a *'membrane'* for the rest of the paper.

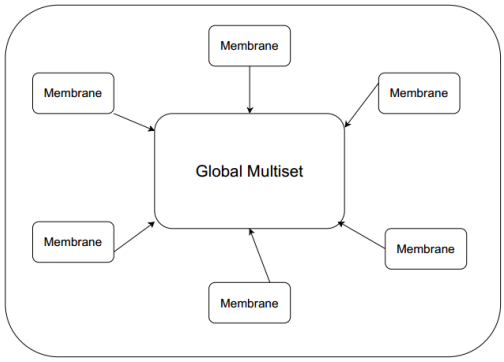

Fig. 1. Membrane Architecture

Before proceeding any further, we must outline the basic architecture utilized to achieve a decentralized execution. Fig 1 represents the overall design used while executing a workflow. The outermost membrane represents the skin membrane, the inner membranes demonstrates the individual workitems. As visible, all membranes pass data and parameters to the global multiset. Each membrane operate on the objects availabile in the multiset locally. Hence, there is no need of a central controlling authority. After completing an execution, the membrane dissolves and leaves the transformed objects in the multiset. This procedure is

followed till the objects are pushed out to the surrounding i.e. the execution of a workflow has completed.

In the membrane computing model, we focus on the property of elastically provisioning resources, by the membranes themselves using cell division rules. An example of cell division rules was outlined previously. Using such type of division rules, the membrane is instantly divided, and the load is dynamically shared between the parent membrane and the newly born child membranes. It should be pointed out here that the provisioning of resources is done autonomously (i.e. by a membrane itself), thereby the need of the cloud service provider to locate and provision resources is removed completely. While provisioning extra resources (resource elasticity property), or in membrane terminology, when a parent membrane is dynamically divided into child membranes, the child membranes also read and wrote to the same multiset. In this way, a membrane is self capable of finding and provisioning resources into existing workflow. Previous works in literature do not take this property into consideration, they either depend on cloud service provider or on the orchestrator to accomplish this functionality. We on the other hand, focus self-provisioning of resources by the membranes themselves.

During execution, when a membrane is dynamically divided, the load has to be shared between parent and child membranes. Since the provisioning of resources, or rather, the membrane division process is autonomous (no orchestrator), therefore one might think that the division process will take some time. However, in the results section we prove that the division process, rather than slowing things down, actually speeds up the process.

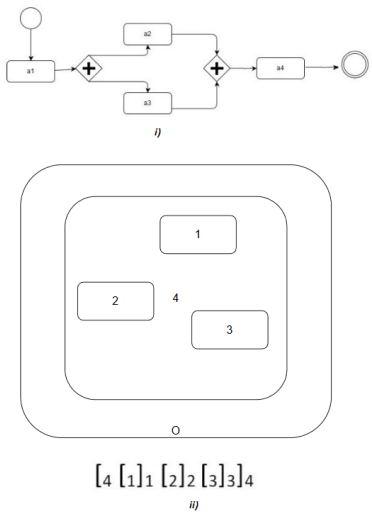

Fig. 2. A Simple Workflow & Membrane Representation

To dynamically divide a membrane, a *load-indicator* is instantiated with each membrane. When a threshold value is reached, for either the response time, the throughput, the queue size etc., the membrane division rule is invoked and the parent membrane is divided into multiple child membranes. Whenever, the load-indicator sensed the load has reduced, the membrane merging rule was invoked and the extra provisioned resources were released.

In the proposed model, the multiset is considered as a stable, semantically distributed shared memory offered as a service. Semantic space allow an inherent capability for parallel processing and a distributed stable data management platform. This type of platform allows huge volumes of data to be stored in a semantic format, with event-driven and asynchronous mode of communication. Such type of a 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. Since, the discussion of semantic spaces is out of scope for this paper, we direct the interested reader to [7], [8].

$$
\begin{aligned}
&[_1 a]_1 \longrightarrow [_2 b,in]_2\ [_3 c,in]_3\ \delta \\
&[_2 b]_2 \longrightarrow [_4 d,in]_4\ \delta \\
&[_3 c]_3 \longrightarrow [_4 e,in]_4\ \delta \\
&[_4 de]_4 \longrightarrow [_o f,out]_o\ \delta
\end{aligned}
$$

Fig. 3. Execution Rules

To demonstrate the procedure used in the membrane computing model to execute a workflow is explained with the help of an example. A simple workflow with four activities in BPMN (Business Process Management Notation) format is demonstrated in Fig 2. The equivalent representation using membranes is shown in the same Figure. The rules to execute the workflow are shown in Fig 3. The rules here do not include any division behavior, rather they specify the execution procedure from a global point of view. It should be pointed out here that the elastic behavior or the cell division rules are specified locally to each membrane.

In this Figure, a,b,c,d,e,f are multisets of objects. The execution begins when membrane one performs a transition, and writes the transformed data to the multiset (Rule 1). Further, using Rule 1 the membrane dissolved (the δ indicator) and let membranes two and three access the multiset to read the objects b and c respectively. A similar procedure is followed for all the execution rules. The last rule demonstrates the completion of transition activities, and the system halts. In the proposed work, we have used the notion of the *dependency operator*, enabling a sense of determinism during execution. In this paper, we propose the use dependency operator as $\xrightarrow[Dependency]{(Details)}$, where Dependency represent the type of dependency (data or control) and Details demonstrate the list of dependent

membranes. It is a known fact that the membrane at same level execute their rules in a non-deterministic manner. However, owing to control dependencies, certain membranes have to be restricted from execution. In such scenarios, the proposed operator provides a certain level of determinism and restrict few of the membranes from execution. For example, consider the following dependency rule:

$$[_1[_{S_1}a]_{S_1}[_{S_2}b]_{S_2}[_{S_3}c]_{S_3}]_1 \xrightarrow[Control]{S_2,S_3} [_1[_{S_1}a]_{S_1}]_1\delta \quad (1)$$

In this rule, the dependency is control and the membrane list contains {*S2, S3*}. The rule implies that prior to the execution of *S1*, or, in other words, prior to the application of this rule, both S2 and S3 must dissolve (or S1 and S2 must have completed their resp. tasks).

It was outlined previously, the membranes are self sufficient to procure resources on their own. To accomplish this functionality, cell division rules are utilized. For example, consider a rule

$$[_1\,a]_1 \longrightarrow [_{1.1}\,a]_{1.1}[_{1.2}\,a]_{1.2}$$

In this example, membrane one is divided into two halves, each having the same processing functionality and capability. The division process is carried out using symport rules, involving the membrane and the load-indicator. As soon the membrane divided, the execution procedure and order became

$$[_{1.1}\,a]_{1.1}[_{1.2}\,a]_{1.2} \longrightarrow [_2 b,\ in]_2\ [_3 c,\ in]_3\ \delta$$

If, in the middle of a transaction, the provisioned resources have to be released, then cell merging rules are invoked, e.g.

$$[_{1.1}\,a]_{1.1}[_{1.2}\,a]_{1.2} \longrightarrow [_1\,a]_1$$

Hence, by applying the division rules and the merging rules, resources can be provisioned and released. The specification and application of these rules is dependent on each individual membrane, there is no authority that controls the division and merging process. The membranes are self-sufficient and self-capable of invoking division rules and merging rules independently. During real world experimentation, we have also used the same procedure to execute the scientific workflows.

In the membrane computing model, each membrane is assigned a specific role. Moreover, the membranes are assigned a unique name and an identifier. Membranes assigned to the same role can execute the same functionality. In the *Future Internet*, the issue of reliability is inevitable, therefore redundant membranes should be kept as back-up in case one fails in the middle of an execution. Next, while executing a workflow there are certain input and output dependencies that must be resolved before proceeding. In the proposed work, these dependencies are specified in an XML format thereby providing a straightforward mapping to a machine readable format. Since, a lot of work [1], [5], [13] has been done to resolve dependencies and automate the execution of traditional workflows, therefore we rely on those procedures to proceed with the execution.

Listing 1.1. ResourcePool

```
<ResourcePool>
<Resource>
<Address>
http://10.200.40.139/Traffic/Diverte/node1
</Address>
<Endpoint>
.
.
</Endpoint>
</Resource>
<Resource>
<Address>
http://10.200.40.132/Traffic/Diverte/node2
</Address>
<Endpoint>
.
.
</Endpoint>
</Resource>
<Resource>
<Address>
http://10.200.40.131/Traffic/Diverte/node4
</Address>
<Endpoint>
.
.
</Endpoint>
</Resource>

</ResourcePool>
```

Now, to begin with the execution, a workflow is specified to the multiset. Every participating membrane read its corresponding dependencies (a low level locking mechanism). In the experiments, we used an XML schema. It should be noted that our motive is *'not'* to introduce a new *description language*. The schema is not limited and was constructed using the principles of domain specific languages. Therefore, any type of workflow can be mapped to a machine readable and executable format, thereby presenting a language independent interface. In that case, each membrane must be equipped to handle any type of description. A question arises here: How do membranes understand these specifications? To interpret these constructs, each membrane is equipped with a local interpreter. Hence, an extra layer is added to the membranes to correctly interpret the workflow description (either XML or normal rules).

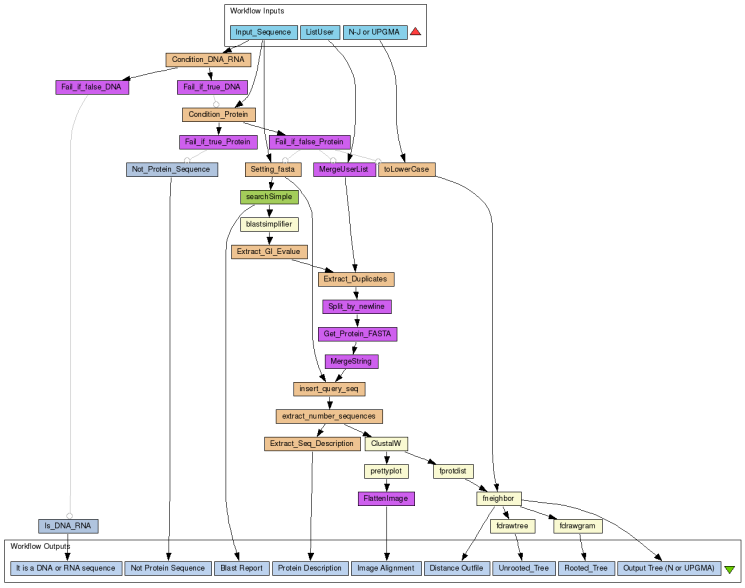

Fig. 4. One of the Workflows for Experimentation

4 Results

4.1 Experimental Setup

In order to evaluate the proposed work, and to check the viability in actual deployment scenarios, we have conducted experiments with multiple workflows collected from myexperiment.org. The execution of these workflows was achieved in 1) Inside the Institute's Intranet 2)Virtual Machines within the Computing Lab of the Institute. The configuration of each machine in the Intranet is i5 Processor with 4 GB RAM, whereas the resource pool had multiple machines, each having Quad-Core processors with 8GBs and 16GBs of RAM. The distributed shared memory, MozartSpaces[3], was deployed as a RESTful service. The application container for the services was Apache Tomcat v7.0.41. The experiments were conducted specifically keeping in mind the capability to provision resources independently, specifically we concentrate on Infrastructure as a Service.

To Study the effect of a low bandwidth environment, the networking capabilities of each Virtual Machine (VM) was constrained. In this paper, one of our motive was to test the model's performance and feasibility to execute a decentralized scientific workflow under duress with limited bandwith capabilities. We discuss the behavior of the model in the next subsection.

[3] http://www.mozartspaces.org/

4.2 Execution Efficiency

As outlined previously, we have chosen workflows from myexperiment. The workflows are uploaded by the people in the research community, and spans different fields viz. Bio-Informatics, Protein Sequence Analysis, nucleotide and protein sequence analysis. Each workflow was executed multiple times. A total of 13, 28, and 18 services were developed for workflow I[4], II[5], and III[6].

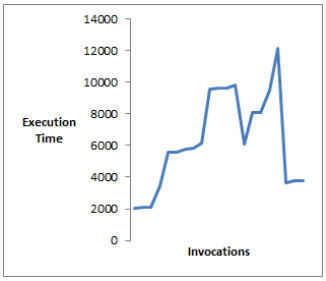

Fig. 5. Execution Time WF-I No Constraints

During the experiments, it was assumed that each of the discrete workitems are realized by a web service. In Fig 5, we have shown the execution time of the workflow I, with no constraints on the bandwidth capabilities. As visible, the execution time of the workflow started at a normal pace. But, when the invocations increased linearly, the execution time followed. The moment, the load-indicator (*of each individual membrane*) sensed duress, a new resource or a Virtual Machine (VM) was provisioned from the resource pool. The newly provisioned resource was made aware of the stable storage location and access methodology.

After provisioning resources, the execution time experienced a sudden drop. This is clearly visible in Fig 5. It should be noted here that in the experiments, each workitem (or membrane) provisioned resources on its own. The first reduction in the execution time is due to the fact that only a few membranes provisioned an extra resource. Therefore, the drop is not that much steep. However, a sudden decrement in the execution time at the end of the graph indicate multiple VMs were provisioned to complete the workflow. We invoked the same workflow multiple times (in regular intervals) so as to test the behavior of autonomously provisioning resources. It should be noted here, when the membranes provisioned extra resources, it happened when the load indicator sensed duress for the *new* incoming request. The already existing requests were not dynamically *migrated* (live migration). In business terminology, the SLAs (Service Level Agreements)

[4] http://www.myexperiment.org/workflows/244.html
[5] http://www.myexperiment.org/workflows/124.html
[6] http://www.myexperiment.org/workflows/1477.html

were violated for the new requests only, there is no need to provision resources for non-SLA violating requests (*principles of cost elasticity*).

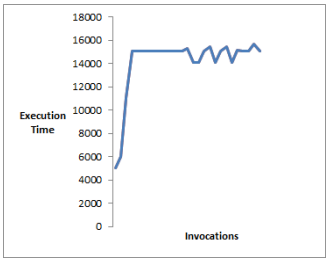

Fig. 6. Execution Time WF-III No Constraints

The same procedure was followed to execute workflows II and III. The result demonstrating the execution time for workflow II is shown in Figure 6. As visible, the execution time dropped the moment the load indicator sensed an increment in the load of the individual membrane. Hence, looking at the execution results for workflows I and II, it can be said that though the membranes provisioned resources autonomously, but the execution efficiency was never compromised. The results demonstrate the feasibility of the membrane computing paradigm towards executing scientific workflows without requiring a centralized controller, or without pre-defined arrangements with the cloud service provider (Infrastructure as a Service).

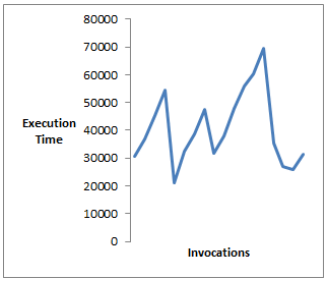

Fig. 7. Execution Time WF-I Limited Bandwidth

In the experiments conducted so far, the bandwidth of each VM was not limited. During experimentation, we limited the bandwidth of each VM deployed to 2KBytes/s. This was done to evaluate the feasibility of the prototype in real world scenarios. The resulting graph for the execution time is shown in Fig 7 (workflow I). As demonstrated in the Figure, the provisioning of extra resources resulted the same sudden drop in the execution time. However, in this case the

execution time increased. This effect is due to the fact that the bandwidth is limited and each individual membrane required some time to receive the data dependencies. Moreover, it was observed that there were instances when the request was dropped due to severe congestion.

Fig. 8. Network Performance Limited Bandwidth

In Figures 8 and 9, we have shown the snapshots of the network perfromance of two VMs chosen at random. It can be seen from the Figures that the network capabilities witnessed its peak during execution. It is at that instant, the new incoming requests were dropped. Further, the performace showed that whenever the load increased beyond the threshold limit, new VMs were provisioned to balance the load.

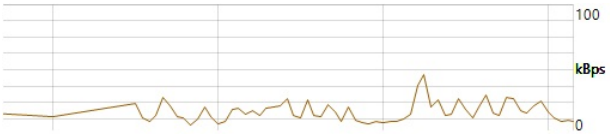

Fig. 9. Network Performance Limited Bandwidth

5 Related Work

In literature, there are lot of techniques available to execute a workflow, either centrally or decentrally. In the decentralized scenario, the services share data and parameters either by passing messages directly or indirectly. However, only a few are in the context of dynamically provisioning resources for scientific workflows with actual deployment.

Nature inspired metaphors have caught some attention lately. Based on these approaches we found two interesting metaphors 1) Chemistry 2) Physics. A decentralized framework to execute workitems, is proposed in [5], [2]. Fernandez *et al* [2] propose executing a workflow using the chemical paradigm. Similar to our work, the authors used a decentralized environment, however, they used a centralized shared memory (hence, the authors suffered from scalability issues). Moreover, they kept the services fixed to execute the workitems, with no provision of dynamic adaptations. Further, the issues of elasticity is not addressed. A

similar method to achieve orcehstration and choreography is proposed in [5]. The author used the same chemistry model to achieve orchestration and choreography. Next, the work in physics, focus achieving *motion co-ordination*, using the notion of '*Computational Fields*' [16]. However, the focus is distributed motion co-ordination, not scientific workflow execution. A similar to technique to synchronize the motion of 'Unmanned aerial system (UAS)' is proposed in [17]. The author has used the notion of physics inspired co-fields to accomplish this functionality. A similar physics inspired technique is proposed in [10]. The authors also focus achieving a decentralized service composition in dynamic environments. However, the focus is not scientific workflow execution.

A cloud based middleware is proposed in [11]. It is a platform proposed to elastically provision resources at run time. However, the main focus of [11] is not scientific workflow execution in a decentralized environment. In [9], a comparison between the resource consumption between a high performance computing machine and a cloud based environment is presented. The cloud experiments are conducted on Amazon's EC2. The author found, the resources provisioned from the cloud were not that powerful as compared to a traditional HPC machine. They found, though executing scientific workflow was acceptable, however the data transfer costs were high. This is one of the factors we will be focusing in the future works. How to find an optimal balance between resource and cost elasticity? [12] introduces the concept of scheduling data intensive scientific workflows in a cloud based environment with virtual clusters. The scheduling is based on the 'iterative ordinal optimization' algorithm. The application of the algorithm produces a significantly lower processing overhead involved in scheduling the workflows. In this paper, we also achieved a decentralized workflow execution based on real virtual clusters.

6 Conclusion and Future Work

In this paper, we introduced the membrane computing paradigm towards realizing a scientific workflow. The membranes acted independently, with a global perspective, and provisioned resources autonomously. We validated our approach over real services, on real virtualization platform within the computer labs of our institute. We were able to demonstrate that the proposed methodology was effective in provisioning resources autonomously at run-time, thereby validating the technique for an actual deployment. Future work in this direction includes work towards extending the methodology towards incorporating mobile devices. Further, we are looking towards deploying the services on a grid based facility, while provisioning resources from a public cloud. Finally, we propose to study the effects of cost elasticity on resource provisioning.

7 Acknowledgement

The authors would like to thank fellow Ph. D students, to help with the technical issues faced during experimentation, and figuring out with the technicalities of

Membrane Computing. We would also like to thank the people in the research community, for sharing their scientific workflows.

References

1. Barker, Adam, Christopher D. Walton, and David Robertson. "Choreographing web services" Services Computing, IEEE Transactions on 2, no. 2 (2009): 152-166.
2. Fernandez Hector, Cedric Tedeschi, 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).
3. Bell, Gordon, Tony Hey, and Alex Szalay. "Beyond the data deluge." Science 323, no. 5919 (2009): 1297-1298.
4. Alonso, Gustavo, Divyakant Agrawal, Amr El Abbadi, and Carl Mohan. "Functionality and limitations of current workflow management systems." IEEE Expert 12, no. 5 (1997): 105-111.
5. 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.
6. Paun, Gheorghe. "Computing with membranes" Journal of Computer and System Sciences 61, no. 1 (2000): 108-143.
7. Wang, Xiaohang, Jin Song Dong, ChungYau Chin, Sanka Ravipriya Hettiarachchi, and Daqing Zhang. "Semantic space: An infrastructure for smart spaces." Computing 1, no. 2 (2002): 67-74.
8. Zhuge, Hai. "Semantic grid: Scientific issues, infrastructure, and methodology." Communications of the ACM 48, no. 4 (2005): 117-119.
9. Juve, Gideon, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P. Berman, and Phil Maechling. "Scientific workflow applications on Amazon EC2." In E-Science Workshops, 2009 5th IEEE International Conference on, pp. 59-66. IEEE, 2009.
10. Ahmed, Tanveer, and Abhishek Srivastava. "Minimizing Waiting Time for Service Composition: A Frictional Approach." In Web Services (ICWS), 2013 IEEE 20th International Conference on, pp. 268-275. IEEE, 2013.
11. Calheiros, Rodrigo N., Christian Vecchiola, Dileban Karunamoorthy, and Rajkumar Buyya. "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds." Future Generation Computer Systems 28, no. 6 (2012): 861-870.
12. Zhang, Fan, Junwei Cao, Kai Hwang, and Cheng Wu. "Ordinal Optimized Scheduling of Scientific Workflows in Elastic Compute Clouds." In Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, pp. 9-17. IEEE, 2011.
13. Zaha, Johannes Maria, Alistair Barros, Marlon Dumas, and Arthur ter Hofstede. "Let?s dance: A language for service behavior modeling." In On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, pp. 145-162. Springer Berlin Heidelberg, 2006.
14. Cecilia, Jos M., Jos M. Garca, Gins D. Guerrero, Miguel A. Martnez-del-Amor, Ignacio Prez-Hurtado, and Mario J. Prez-Jimnez. "Simulating a P system based efficient solution to SAT by using GPUs." The Journal of Logic and Algebraic Programming 79, no. 6 (2010): 317-325.

15. Nishida, Taishin Y. “An approximate algorithm for NP-complete optimization problems exploiting P systems.” In Proc. Brainstorming Workshop on Uncertainty in Membrane Computing, pp. 185-192. 2004.
16. Mamei, Marco, Franco Zambonelli, and Letizia Leonardi. “Distributed motion coordination with co-fields: A case study in urban traffic management.” In Autonomous Decentralized Systems, 2003. ISADS 2003. The Sixth International Symposium on, pp. 63-70. IEEE, 2003.
17. Reza, Hassan, and Kirk Ogaard. “Modeling UAS swarm system using conceptual and dynamic architectural modeling concepts.” In Conceptual Structures for Discovering Knowledge, pp. 331-338. Springer Berlin Heidelberg, 2011.