High Availability-Aware Optimization Digest for Applications Deployment in Cloud

Manar Jammal
ECE Depratment
Western University
London ON, Canada
mjammal@uwo.ca

Ali Kanso
Ericsson Research
Ericsson
Montreal Canada
ali.kanso@ericsson.com

Abdallah Shami
ECE Depratment
Western University
London ON, Canada
ashami2@uwo.ca

***Abstract*—Cloud computing is continuously growing as a business model for hosting information and communication technology applications. Although on-demand resource consumption and faster deployment time make this model appealing for the enterprise, other concerns arise regarding the quality of service offered by the cloud. One major concern is the high availability of applications hosted in the cloud. This paper demonstrates the tremendous effect that the placement strategy for virtual machines hosting applications has on the high availability of the services provided by these applications. In addition, a novel scheduling technique is presented that takes into consideration the interdependencies between applications components and other constraints such as communication delay tolerance and resource utilization. The problem is formulated as a linear programming multi-constraint optimization model. The evaluation results demonstrate that the proposed solution improves the availability of the scheduled components compared to OpenStack Nova scheduler.**

***Index Terms*—High availability, cloud applications, delay tolerance, scheduling algorithms, virtual machines, failure scope.**

I. Introduction

Nowadays, the cloud is becoming the lifeblood of most telecommunication network services and information technology (IT) software applications [1] [2]. With the development of the cloud market, cloud computing can be seen as an opportunity for information and communications technology (ICT) companies to deliver communication and IT services over any fixed or mobile network with high performance and secure end-to-end quality of service (QoS) for end users [3]. Although cloud computing provides benefits to different players in its ecosystem and makes services available anytime, anywhere and in any context, other concerns arise regarding the performance and quality of the services offered by the cloud. One major concern is high availability (HA) of applications hosted in the cloud. Since these applications are hosted by virtual machines (VMs) residing on physical servers, their availability depends on that of the hosts [4] [5] [6]. When a hosting server fails, its VMs and their applications become inoperative. The absence of application protection plan has a tremendous effect on business continuity and IT enterprises. According to Aberdeen Group, the cost of one hour of downtime is \$74,000 for small organizations and \$1.1 million for larger ones [7]; excluding reputation damage that can be significantly greater in the longer term. In addition, the Ponemon Institute study shows that 91% of data centers have experienced unplanned outages in 2011 and 2012 [8]. The solution to these failures is to develop a highly available system that protects services, avoids downtime and maintains business continuity.

High availability is an interesting concept that has attracted several recent research studies. However, the way to attain a certain availability baseline when scheduling VMs or applications changes from one research study to another. In [9], the authors propose a placement approach to generate VM configurations while maintaining high availability for multi-tier applications and improving their performance. They develop a replication strategy to maintain HA constraints based on the mean time between failures (MTBF) while satisfying latency demands to minimize performance degradation for each application. In [10], the authors address the resource allocation problem for deployment of multitier applications. They aim to maximize total service level agreement (SLA) profit while maintaining a certain level of availability. They develop a non-linear programming model to achieve their objective while guaranteeing a level of availability based on a load-sharing fault-tolerance arrangement. Another attempt that maximizes service availability by providing a failure-resiliency plan is proposed in [11]. In [12], the authors address applications components scheduling on a cloud infrastructure. They propose a multi-objective scheduling approach that tends to maximize resource utilization, minimize the cost of application runtime and maximize applications availability through a component replication approach.

Although these solutions try to maximize the cloud-service availability, each approach focuses only on a few aspects of availability. Some attempts tend to maximize availability without considering redundancy models, while others ignore the impact of mean time to failure (MTTF) or interdependency relations and their associated constraints. To address these inadequacies, a scheduling solution is required that considers all the factors affecting availability. This paper aims to demonstrate the effect of applications placement strategy on the HA of the services provided by the virtualized cloud to its end users. To attain this objective, the cloud and the application models have been captured in a unified modelling language (UML) model. Also, we propose a novel scheduling technique that looks into the interdependencies and redundancies between applications components, their failure scopes, their communication delay tolerance and resource utilization requirements. The technique examines not only MTTF to measure the component downtime and consequently its availability, but the analysis is based on the mean time to repair (MTTR), recovery and outage tolerance times as well. This scheduling is modeled as a mixed integer linear programming (MILP) problem.

The main contributions of this work are to:

- Capture all the functionality and availability constraints that affect application placement.
- Reflect availability constraints not only by failure rates of applications components and scheduled servers, but also by functionality requirements, which generate anti-location and co-location constraints.
- Consider various interdependencies and redundancies among applications components.
- Examine multiple failure scopes that might affect the component itself, its execution environment and its dependent components.

This paper is organized as follows. Section II describes the system UML model, including cloud and application tree structures. Section III defines the research methodology for developing the MILP model. Section IV describes the simulation environment and the results of this work. Finally, future work and conclusions are presented in Section V.

II. System Modelling and Schematization

Cloud schedulers that are agnostic of the intricacies of the tenants application may result in suboptimal placements. In these placements, redundant components may be placed too close to each other rendering their existence obsolete because a single failure can affect them all, or the delay constraints can be violated, hindering application functionality. The HA-aware scheduling in the cloud must consider details of both the applications and the cloud infrastructure. We define a different approach where we start by modeling the applications with their functional and non-functional requirements. Then we consider the cloud infrastructure model to be a constrained solution space where we do the mapping between applications, VMs, and servers to maximize the availability. To this end, we modelled the cloud and the application using a unified modelling language (UML) class diagram, as shown in Fig. 1. With this model, we started with a specific cloud infrastructure and a set of requested applications then we ended up by generating an interface between the provider and user side using VM mappings.

A. Cloud Provider Model

The proposed cloud architecture is captured in the UML class diagram. At the root level, the cloud consists of data center networks distributed across various geographical areas. Each data center consists of multiple racks communicating through aggregated switches. Each rack has a set of shelves housing a large number of servers, which can have different capacities and failure rates. Servers residing on the same rack are connected with each other through the same network device (the top of the rack switch). Finally, the VMs are hosted on the servers. This tree structure determines the network delay constraints and consequently the delay between the communicating applications. This architecture divides the cloud into five different latency zones, which will be further discussed in Section III.

In the proposed tree structure, each node *i* has its own failure rate (λ) and MTTR. The MTTF and MTTR parameters divide the intra- and inter- data center networks into availability zones. The availability *avail* of the host *h* depends not only on its λ and MTTR, but on that of corresponding DC and rack *R* as well. Each host *h* can be seen as *(DC, R, S)*. In each zone, the highly available host is selected as follows:

$$avail_h = \frac{MTTF_h}{MTTF_h + MTTR_h} \tag{1}$$

$$where \begin{cases} MTTF_h = \frac{1}{\lambda_{DC}+\lambda_R+\lambda_s} \\ \\ MTTR_h = MTTR_{DC} + MTTR_R + MTTR_S \end{cases}$$

B. Applicationr Model

Applications are typically developed using a component based architecture where each application is made up of one or more components. The application combines its components$'$ functionalities to provide a higher level of service [13] [14]. To maintain availability requirements, each component can have one or more redundant components. The primary component and its redundant ones are grouped into a dynamic redundancy group. In this group, each component is assigned a specific number of active and standby redundant components. As shown in the UML model, each redundancy group is assigned to at most one application, which consists of at least one redundancy group.

As for the component, it belongs to one component type. A component type represents an executable software deployment. From this perspective, the component represents a running instance of the component type. Components of the same type have the same attributes defined in the component type class, such as computational resources (CPU and memory) attributes. Each component can be configured to depend on other components. The dependency relation is captured at the type level and can be configured using the delay tolerance, outage tolerance and communication bandwidth attributes. The delay tolerance determines the minimum required latency to maintain communication between sponsor and dependent components. As for the outage tolerance or tolerance time, it is the time that the dependent component can tolerate without the sponsor one. The same association is used to describe the relation between redundant components that need to synchronize their states.

Finally, each component type is associated with at least one failure type. The list of failure types determines the failure scope of each component type, its MTTF, MTTR, and recommended recovery.

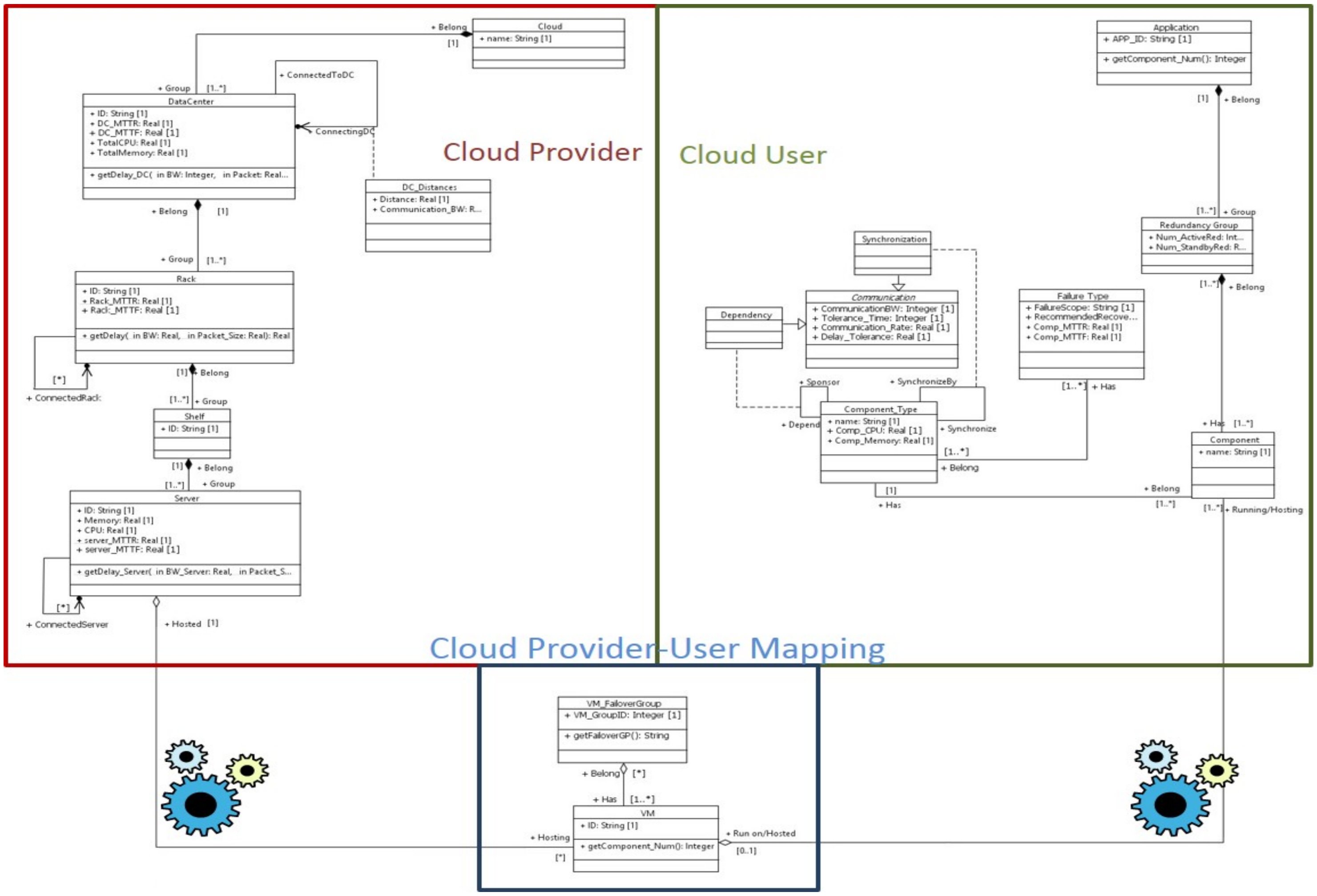

Fig. 1: Cloud-Application UML Class Diagram.

C. *VM Mapping*

Each component of the application model is scheduled on a server in the cloud provider model using VM mappings. Each VM can be hosted on one server and can have at least one component instance running in it. Sudden failure events can occur to cloud-application such as natural disaster, network or runtime failures [15]. In order to deal with these events, the inoperative VMs are switched off, and a failover group takes over the control. The failover group consists of at least one VM, which is a redundant VM of the inoperative one.

As mentioned earlier, the proposed HA-aware scheduling technique searches for the optimum physical server to host the requested component. Whenever a server is scheduled, a VM is mapped to the corresponding component and to the chosen server. Therefore, a component can reside on that VM.

III. High Availability Model

The VM placement solution generates mappings between the cloud physical servers and VMs on which the tenants applications are hosted while satisfying various constraints.

The HA-aware scheduling technique provides an efficient and highly available allocation by satisfying the following constraints:

1) *Capacity Constraints:* These are functional constraints, which are satisfied by searching for servers that meet the resource needs of each application. In the proposed model, the computational resources consist of the CPU and memory.
2) *Network Delay Constraints:* Using these constraints, another list of servers is generated. These servers satisfy the latency requirements to avoid service degradation between communicating applications. It is assumed that the delay requirements are divided into five delay types (i.e., latency zones) as follows:
   a) D_0 *Type:* Requires that all communicating components should be hosted on the same VM and consequently on the same server.
   b) D_1 *Type:* Requires that all the communicating components should be hosted on the same server.
   c) D_2 *Type:* Requires that all the communicating components should be hosted on the same rack.
   d) D_3 *Type:* Requires that all the communicating

components should be hosted on the same DC.

e) *D_4 Type:* Requires that all the communicating components can be hosted across different data centers but must be within the same cloud.

3) *Availability Constraints:* These constraints prune the candidate servers generated by the capacity and delay constraints to select the ones that maintain a high level of application′s availability. In order to maximize the HA of an application, three sub-constraints should be satisfied:

a) *Failure Rate Constraint:* It determines that the selected server should maximize component′s availability. In order to satisfy this constraint, the model searches for the server with maximum MTTF and minimum MTTR. Whenever it is found, then the $MTTF_c^A$ of the component *C* after being hosted can be calculated as follows:

$$MTTF_c^A = \frac{1}{\lambda_c + \lambda_h} \tag{2}$$

b) *Dependency Constraint:* This constraint is divided into two sub-constraints:
Co-location Constraint: This is valid whenever the tolerance time of the dependent component is lower than the recovery time of its sponsor. When the dependent component cannot tolerate the absence of its sponsor, then the failure of its server, its sponsor or sponsor′s server affects it. In order to minimize its failure rate, both dependent and sponsor should share same server.
Anti-location Constraint:: It requires that the dependent component and its sponsor should be placed on different servers. This is valid whenever the tolerance time of the dependent component is greater than the recovery time of its sponsor. By considering this case, the MTTF of the application will be maximized because of its inverse proportionality relation with λ.

c) *Redundancy Constraint:* It basically prevents redundant components of a primary one from residing on the same server and requires that they should be placed far away from each other as the delay constraints allow.

With these constraints, we developed a MILP model that minimizes components′ downtime while finding the optimal physical server to host them.

A. Mathematical Formulation

This section introduces a MILP model to solve the HA-aware placement problem. The proposed MILP model was solved using the IBM ILOG CPLEX optimization solver.

1) Notations

Various parameters were used to solve the placement problem and develop the MILP model.

a) Input Paramters

Let a virtual machine be denoted as *V* and a server as *S*. Each VM consists of an application $\{A\}$, which consists of a

Notation	Significance	Representation
R	Resource Type: CPU or memory	Number of Cores and MB of RAM
$RED_{cc'}$	Redundancy matrix of C and C'	{0,1}
$DEP_{cc'}$	Dependency matrix of C and C'	{0,1}
$DEL_{ss'}$	Delay between S and S'	second
OT_c	Outage tolerance of C	hour
RT_c	Recovery time of C	hour
DT_c	Delay tolerance of C	hour

TABLE I: Variable Notations

specific number of components $\{C\}$, which are of component types $\{CT\}$. Therefore, each application is a set of *C* and *CT* and can be denoted as *A* =$\{$*C, CT*$\}$. This notation ensures that whenever a set of components *C* of types *CT* is scheduled, its corresponding application is considered hosted.
As for the computational resources, L_{cr} and L_{sr}^T denote the set of resources, which can be memory or CPU of component and server respectively. Table 1 shows the various parameters notations used in the MILP model.

b) Decision Variables

The decision variables are defined as follow:

$$X_{cs} = \begin{cases} 1 & if\ S\ host\ C \\ 0 & otherwise \end{cases} \tag{3}$$

$$z_c = \begin{cases} 1 & if\ DEL_{ss'} \leq DT_c \\ 0 & otherwise \end{cases} \tag{4}$$

2) MILP Model

The downtime represents a duration during which a system is unavailable or fails to function. In this paper, the system can be a component or a host. However, the downtime of *C* does not only depend on the component itself *(Downtime$_c$)*, but on its hosting server *(Downtime$_s$)* as well. In order to minimize the overall downtime of *C*, the objective function of the formulated MILP model should minimize $Downtime_c$ and $Downtime_s$. The objective function and its constraints are formulated as follows:

Objective Function:

$$\min \sum_c \sum_s (Downtime_c + Downtime_s) \times X_{cs}$$

Subject to:

Capacity Constraints:

$$\sum_c (X_{cs} \times L_{cr}) \leq L_{sr}^T \qquad \forall\, s,\ r \tag{5}$$

$$\sum_s X_{cs} = 1 \qquad \forall\, c \tag{6}$$

$$X_{cs} \in \{0,1\} \qquad \forall\, c, s \tag{7}$$

Network Delay Constraints:

$$(X_{c's'} \times DEL_{ss'} - DT_c) \leq M \times z_{c'} \quad \forall\, c, c',\ s, s' \qquad (8)$$

$$X_{cs} - 1 \leq M \times (1 - z_{c'}) \quad \forall\, c, c',\ s \qquad (9)$$

$$z_{c'} \quad \in \quad \{0, 1\} \quad \forall\, c' \qquad (10)$$

Redundancy Constraint:

$$X_{cs} + X_{c's} \leq 1 \quad \forall\, c, c', s,\ RED_{cc'} \qquad (11)$$

Dependency Anti-location Constraint:

$$X_{cs} + X_{c's} \leq 2 \quad \forall\, c, c', s,\ DEP_{cc'} \qquad (12)$$

Dependency Co-location Constraint:

$$X_{cs} + X_{c's} \leq 1 \quad \forall\, c, c', s,\ DEP_{cc'} \qquad (13)$$

Boundary Constraint:

$$Downtime_c, Downtime_s \geq 0 \quad \forall\, c, s \qquad (14)$$

As discussed earlier, the HA-aware placement of the application is affected by capacity, delay and availability constraints. Regarding capacity constraints, constraint (5) ensures that the requested components resources must not exceed the available resources of the selected destination server. Constraint (6) determines that the component can be placed on at most one physical server. Constraint (7) ensures that the decision variable *(X_{cs})* is a binary integer.
The delay constraints (8), (9), and (10) ensure that communicating components will be placed on a server that satisfies the required latency. These constraints are applied on the dependency and redundancy communication relations between scheduled components.
The availability constraint (11) reflects the anti-location constraint between a component and its redundant ones. Using constraint (12), the dependent components should share the same server in case their outage tolerance is smaller than the recovery time of their sponsor component. The anti-location constraint between dependent and sponsor components is active in the contrary case as shown in (13). The boundary constraint (14) specifies real positive values for downtimes of *C* and *S*.

IV. MILP Evaluation

To assess the MILP model, evaluations were conducted using different data sets. The MTTF, MTTR and recovery time were used as measures of the downtime and availability of various components. To clarify the importance of the proposed model, it was compared to OpenStack Nova scheduler algorithm [16].

A. Availability Analysis

As mentioned earlier, the availability $avail_c$ of a component *C* is inversely proportional to its downtime. Since the downtime was generated in terms of hours per year, then the availability is calculated as follows:

$$avail_c = (\frac{8760 - downtime_c}{8760} \times 100) \qquad (15)$$

As for the downtime, it changes with delay types and can be calculated in terms of λ, MTTR and/or *RT* of a component, its corresponding *DC*, rack *R* and server *S*. Since our work considers redundancy and failover solutions, then downtime depends on λ and *RT*. For instance, the downtime in D_4 and D_2 type is calculated as (16) and (17) respectively:

$$downtime_c^{D_4} = (\lambda_c + \lambda_s + \lambda_{DC} + \lambda_R) \times RT_c \qquad (16)$$

$$downtime_c^{D_2} = ((\lambda_c + \lambda_s) \times RT_c) + (\lambda_R \times RT_R) + (\lambda_{DC} \times RT_{DC}) \qquad (17)$$

B. Computational Complexity

Any scheduling problem can be defined as a triplet $\alpha \mid \beta \mid \gamma$. Starting with α, it represents the problem environment. As for β and γ, they represent the problem constraints and the objective to be optimized respectively [17]. This triplet can take various fields depending on the scheduling type. Since the proposed scheduling work has *n* components to be assigned to *m* servers while minimizing downtime, it can be formulated as special case of transportation problem. It can be represented as $Q_m \mid p_j \mid \sum h_j(C_j)$, where Q_m indicates that the problem environment consists of *m* different parallel machines, p_j determines that a job *j* can be processed using one machine *m* and finally *h(k)* represents the cost function to be optimized. This special case is known as the bipartite matching or assignment problem. It is represented as bipartite graph *G = (n_1, n_2, a)*. This graph consists of two nodes n_1 and n_2 connected using arc *a*. This arc *a* = {*j*, *k*} assigns node *j* of set n_1 to node *k* of set n_2 and is represented by the decision variable X_{cs} defined in Section III. This type of scheduling problems is formulated using linear programming models, but it is characterized by NP-hard complexity hierarchy. Because of the NP-hardness of the proposed optimization model, it is only feasible for small DC networks [18]. In the evaluated small network, the number of variables generated in the optimization solver is approximately 4000.

C. OpenStack Filter Scheduler

OpenStack is an open source cloud management system that satisfies the needs of private and public clouds using computing, networking and storage services [16]. Nova is one of the computing components of OpenStack that schedules VM based on a predefined instance type such as CPU or RAM. Nova compute service provides different types of filters and weights to host an instance. During the filtering stage, the scheduler generates list of servers that are capable of hosting the instance. Then weight and cost functions are applied to this list to determine the best compute node for it [19]. Nova supports other filters that can be used to support HA placement such as availability zone, affinity and anti-affinity filters, however, these existing filters are agnostic of the delay tolerance and inter-VM dependencies, which means they need to be extended. Also, the users have to manually define the needed filter based on the instance properties [19]. Alternatively, our proposed work eliminates the user interference. It provides an automated process for deploying

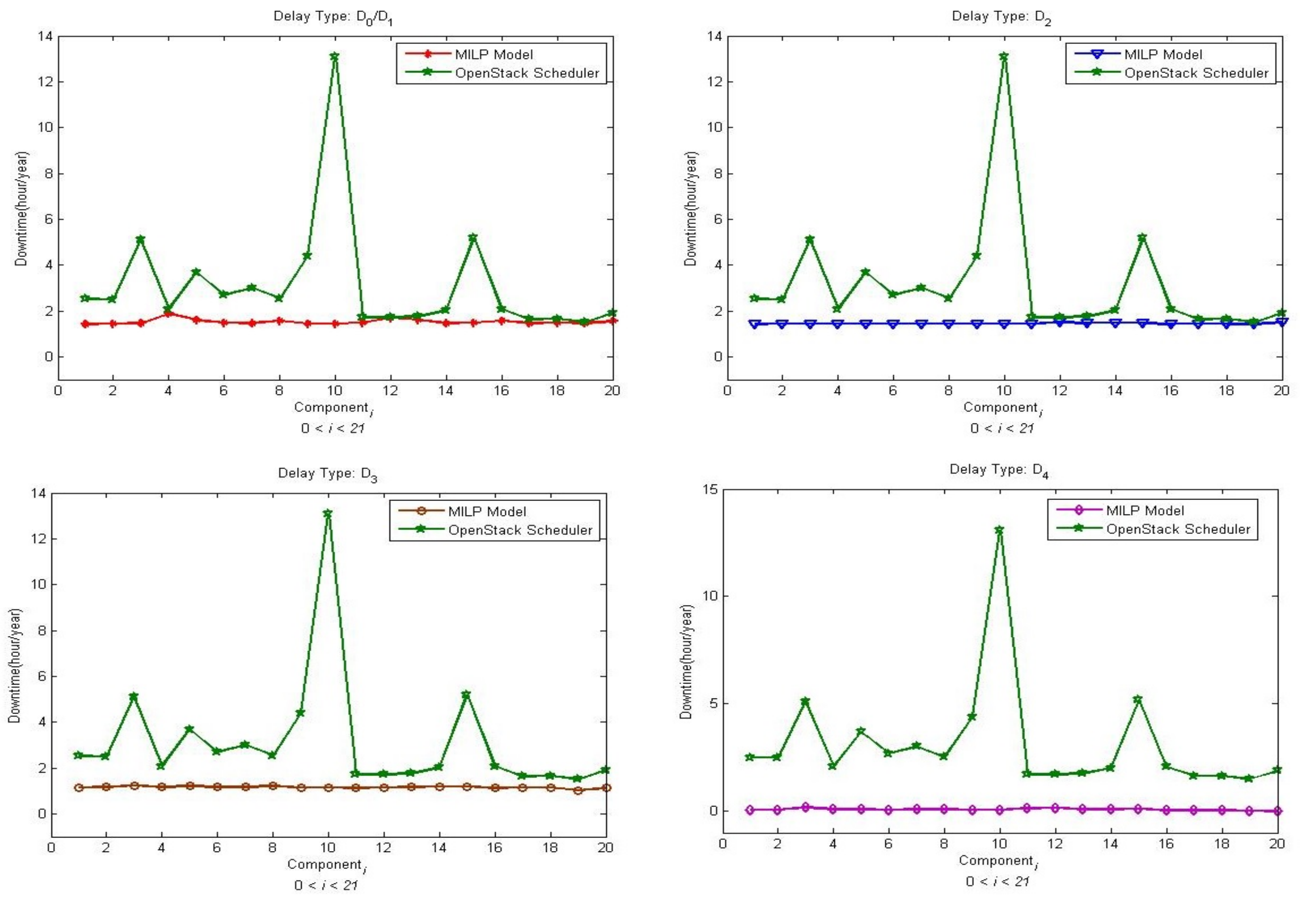

Fig. 2: MILP vs OpenStack Scheduler.

VMs by considering functionality (resources and delay) and availability (different types of dependencies) requirements.

D. Results

Both the optimization model and the OpenStack scheduler were evaluated on a network consisting of 20 components, 2 DCs, 4 racks and 50 servers. The server and component MTTFs were generated using an exponential distribution with mean = 2000 hours for both [20]. As for the server and component MTTR, a normal distribution was used with mean = 3 and 0.05 hours and standard deviation = 1 and 0.016 hours respectively [20] [21]. For server and component recovery times, a normal distribution was used with mean = 0.05 and 0.008 hours and standard deviation = 0.016 and 0.002 hours respectively. To evaluate the interdependencies and redundancies between components, the proposed MILP was evaluated on two real-time Web applications. The Web applications include two types of dependencies among the components: (1) the synchronization dependency between the active component and its replicas, and (2) the functional dependencies where the App server depends on database server and sponsors the HTTP server.

1) MILP vs OpenStack Nova Scheduler

The MILP model was compared to the core and RAM filters in Nova scheduler in order to show the impact of availability constraints on the downtime of components per year. Using these filters, only servers with sufficient RAM and CPU cores are eligible for hosting VMs. Fig. 2 shows the downtime difference between MILP model and OpenStack scheduler for different delay zones. Since these filters do not consider delay requirements, they generated similar results for all delay types. As the delay zones widen the solution space for the components placement, the difference between OpenStack scheduler and MILP increases gradually. Using the HA-aware MILP model, the downtime of component is reduced by 35%, 39% and 52% for D_0/D_1, D_2 and D_3 types respectively. As for D_4 type, it allows scheduling component on any server in the cloud and allows placing primary and redundant components on two different DCs. In this case, if the server, rack or DC of component C fails, C will be down until it fails over to its redundant. Therefore, the downtime of a C is calculated using (16). But sometimes a sponsor component can affect the downtime of its dependents if the latter cannot tolerate its failure. In any case, the downtime is reduced significantly by 97% using the HA-aware technique in D_4 type.

2) Availability Improvement within MILP Model

Although the MILP model maximizes the availability of components, its results change among different delay zones. Since D_0 and D_1 types require placement of components in the

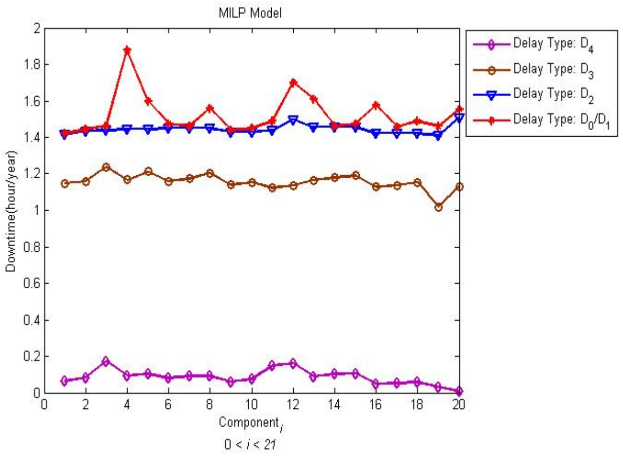

Fig. 3: Availability Improvement among Delay Zones.

same server, then the solution space of the selection process is limited. Consequently, the availability is affected and depends on the recovery time of the hosting server, rack and DC. However, the availability is highly improved in D_4 because it depends only on the recovery time of the hosted component. Additionally, this delay zone eliminates the restrictions on the locations of the components. Fig. 3 shows the difference between downtime among the 5 delay zones. Compared to D_4 results, the availability of components is reduced as more restrictions are added on the servers location.

V. Conclusion

Unexpected cloud-services outages can have a profound impact on business continuity and IT enterprises. The key to achieving availability requirements is to develop an approach that is immune to failure while considering real-time interdependencies and redundancies between applications. This paper has addressed the problem environment from different vantage points to generate highly available optimal placement for the requested applications. The proposed MILP model minimizes the downtime of applications, but its computational complexity limits the evaluation on large networks. Therefore, this model will be associated with a heuristic solution in the future work. The heuristic will solve the scheduling problem in polynomial time while satisfying all QoS and SLA requirements and differentiating between mission critical and standard applications. Also, the proposed approach will be extended to include multiple objectives such as maximizing the HA of applications′ components and maximizing resource utilization of the infrastructure used.

Acknowledgment

This work is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC-STPGP 447230) and Ericsson Research. The authors also would like to thank Philip Kurowski for his contribution in the comparison with OpenStack Scheduler.

References

[1] M.A. Sharkh, M. Jammal, A. Shami, and A. Ouda, "Resource allocation in a network-based cloud computing environment: design challenges," *IEEE Communications Magazine,* vol. 51, no.11, pp. 46-52, Nov. 2013.

[2] Microsoft, "The Economics of the Cloud," http://www.microsoft.com/en-us/news/presskits/cloud/docs/the-economics-of-the-cloud.pdf, Nov. 2010.

[3] ITU, "Cloud Computing Benefits from Telecommunication and ICT Perspectives," http://www.itu.int/dms_pub/itu-t/opb/fg/T-FG-CLOUD-2012-P7-PDF-E.pdf, Feb. 2012.

[4] L. Grit, D. Irwin, A. Yumerefendi, and A. Chase, "Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration," *in 2nd International Workshop on Virtualization Technology in Distributed Computing,* 2006.

[5] D. Jayasinghe, C. Pu, et al., "Improving Performance and Availability of Services Hosted on IaaS Clouds with Structural Constraint-Aware Virtual Machine Placement," *in IEEE International Conference on Services Computing (SCC),* July 4-9, 2011, pp.72-79.

[6] J. Dean and L. Barroso, "The Tail at Scale," *Communications of the ACM,* vol. 56, no. 2, pp. 74-80, Feb. 2013.

[7] Aberdeen Group, "Why Mid-Sized Enterprises Should Consider Using Disaster Recovery-as-a-Service," http://www.aberdeen.com/Aberdeen-Library/7873/AI-disaster-recovery-downtime.aspx, April 2012.

[8] Ponemon Institute, "2013 Study on Data Center Outages," http://www.emersonnetworkpower.com/documentation/en-us/brands/liebert/documents/white%20papers/2013_emerson_data_center_outages_sl-24679.pdf, Sep. 2013.

[9] G. Jung, K.R. Joshi, M.A. Hiltunen, R.D. Schlichting, and C. Pu, "Performance and availability aware regeneration for cloud based multitier applications," *in IEEE/IFIP Conference Dependable Systems and Networks,* June 28-July 1 2010, pp. 497-506.

[10] B. Addis, D. Ardagna, B. Panicucci, and L. Zhang, "Autonomic Management of Cloud Service Centers with Availability Guarantees," *in IEEE International Conference Cloud Computing (CLOUD),* July 5-10, 2010, pp. 220-227.

[11] A. Abouzamazem and P. Ezhilchelvan, "Efficient Inter-cloud Replication for High-Availability Services," *in IEEE International Conference Cloud Engineering (IC2E),* March 25-27, 2013, pp. 132-139.

[12] M.E. Frincu and C. Craciun, "Multi-objective Meta-heuristics for Scheduling Applications with High Availability Requirements and Cost Constraints in Multi-Cloud Environments," *in IEEE Conference Utility and Cloud Computing,* Dec. 5-8, 2011, pp.267-274.

[13] SA Forum, "Service AvailabilityTM Forum Application Interface Specification," http://www.saforum.org/ResourceCenter/Download/16627~333319?view=1, June 2014.

[14] P3 InfoTech Solutions, "Web Application Deployment in the Cloud Using Amazon Web Services From Infancy to Maturity," http://www.slideshare.net/p3infotech_solutions/web-application-deploymentaws#, July 2013.

[15] P. Bodik, F. Armando, M.J. Franklin, M.I. Jordan, and D.A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," *in ACM Symposium Cloud Computing,* June 10-11, 2010, pp. 241-252.

[16] OpenStack, "OpenStack Cloud Software," http://openstack.org.

[17] M. Pinedo, *Deterministic Models: Preliminaries, Scheduling Theory, Algorithms and Systems,* Spring, New York, pp.13-33, 2008

[18] O. Kone, C. Artigues, P. Lopez, and M. Mongeau, "Event-based MILP models for resource-constrained project scheduling problems," *Computers and Operations Research,* vol. 38, pp. 313, Jan. 2011.

[19] OpenStack, "Filter Scheduler," http://docs.openstack.org/developer/cinder/devref/filter_scheduler.html#costs-and-weights, Feb. 2013.

[20] Reliability HotWire, "Availability and the Different Ways to Calculate It," http://www.weibull.com/hotwire/issue79/relbasics79.htm, Sep. 2007.

[21] EventHelix, "System Reliability and Availability," http://www.eventhelix.com/realtimemantra/faulthandling/system_reliability_availability.htm#.U-AcuPldXCf, 2014.