Disaster Recovery for Cloud-Hosted Enterprise Applications

Long Wang, Richard E Harper, Ruchi Mahindru, and Harigovind V Ramasamy

IBM T. J. Watson Research Center
Yorktown Heights, NY, USA
{wanglo, reharper, ruchi, hvramasa}@us.ibm.com

***Abstract*: We describe disaster protection and recovery of cloud-hosted enterprise applications both at the cloud infrastructure level and at the application level. We explore scenarios which favor one option over the other, and scenarios where a combination of both are required for effective and end-to-end protection. Through case studies grounded in the experience of implementing disaster recovery for IBM's Cloud Managed Services (CMS) platform, we highlight the complexities of protecting enterprise applications on the cloud. For recovery planning and execution, we present a scheduling algorithm that recovers machines hosted on cloud by taking into account application-level logical dependencies and the business criticalities of the applications.**

Keywords: disaster recovery, disaster protection, recovery scheduling, recovery planning

I. Introduction

Cloud systems are popularly used for hosting enterprise applications because cloud systems provide rapid provisioning, convenient deployment and simplified management of computing resources and applications with pay-as-you-go pricing models. In particular, platform-as-a-service (PaaS) clouds [34] provide a platform for developing, running, and managing applications. They automate the deployment and configuration of middleware (e.g., databases and messaging queues) and run-time environments required for developing and deploying those applications. For example, database-as-a-service is a PaaS cloud service, which frees cloud users from installing or configuring their own databases. PaaS cloud services have built-in elasticity, which frees the application developer from the need to reserve large amount of resources to handle the occasional peak loads. Examples of PaaS include IBM Bluemix [5], Google App Engine [6], Microsoft Azure SQL database [7], and SalesForce Heroku [8].

Maintaining continuity of information technology (IT) operations despite the occurrence of disasters is critical for many businesses. The disasters may be natural disasters such as earthquakes or hurricanes, or man-made disasters such as terrorist attacks or power outages. *Disaster recovery (DR)* refers to the ability to recover from outages caused by disasters. A number of industries (e.g. financial institutions and health care providers in the United States) are bound by law to recover from IT outages caused by disasters within specified time intervals. With the increasing popularity of cloud platforms to host business applications, DR has become an important service offered by providers of enterprise-class cloud systems, including those of PaaS clouds.

In this paper, we focus on the DR of cloud-hosted enterprise applications. IBM *Cloud Managed Services (CMS)* platform [4] is IBM's flagship enterprise cloud platform, and offers managed services both at the IaaS level and at the PaaS level. Through case studies grounded in the experience of implementing DR for SAP [31] and Oracle [32] applications on the IBM CMS platform, we highlight the complexities of protecting enterprise applications on the cloud. For recovery planning and execution, we present a scheduling algorithm that recovers cloud machines by taking into account application-level logical dependencies among the machines and the business criticalities of the applications resident on those machines.

In previous work [1], we presented challenges and experiences with providing DR for enterprise-class clouds at the IaaS level. In this paper, we broaden the scope to include both PaaS and IaaS clouds. We describe options for disaster protection and recovery of cloud-hosted enterprise applications both at the cloud infrastructure level and at the application level. We explore situations which favor one option over the other, and situations where a combination of both are required for effective and end-to-end protection.

II. Reference Architecture for Disaster Recovery Of Cloud-Hosted Enterprise Applications

Figure 1 shows a reference architecture for DR of cloud-hosted enterprise applications. The figure depicts an enhanced version of the reference architecture we presented in [1], which focused on infrastructure-level DR. In the cloud context, Khalidi [30] defines a fault domain to be "a set of hardware components – computers, switches, and more – that share a single point of failure." For the purposes of DR, a fault domain is generally a cloud site.

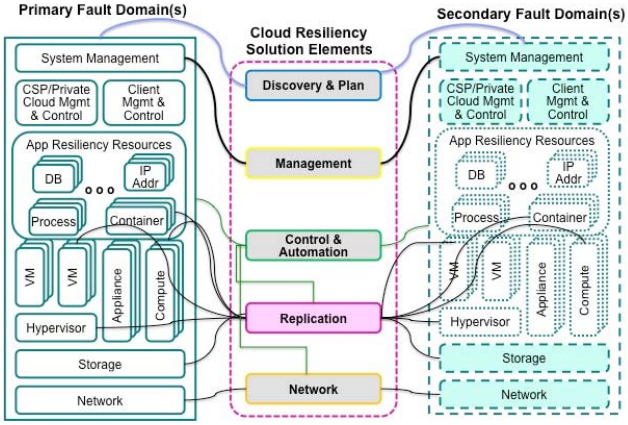

Figure 1: Reference Architecture

We identify five key elements in a DR solution for cloud-hosted enterprise applications: *Discovery and Plan, Control and Automation, Management, Replication,* and *Networking*. *Discovery and Plan* components collect and extract information and characteristics of the cloud infrastructure, services and workloads. *Replication* components replicate data from primary cloud site to the target DR site so that

2159-6190/16 $31.00 © 2016 IEEE
DOI 10.1109/CLOUD.2016.62

workloads can continue execution on the DR site even if an outage brings down the primary site. Replication components may operate at the host-level (e.g., virtual machine level replication), storage level (e.g., block-level replication of all disk updates), service level, middleware level, or application level. *Management* components recover system management capabilities of the primary cloud site at the DR site. System management capabilities include patching, monitoring, authentication, identity services, configuration services, anti-virus, backup, and service request/ticketing. Managed services on the IBM CMS platform [4] implement such capabilities. *Networking* components facilitate data replication and reconfigure the network at the DR site. *Control and Automation* capabilities include a portal for managing all aspects of the DR solution and launching all DR lifecycle activities such as enrolling clients, pre-staging a client environment, DR test, protection, failover, and failback. Control and Automation components are also responsible for infrastructure provisioning, monitoring for failures, orchestration and automation of failure recovery, and triggering adaptive network reconfiguration in response to DR declaration.

An effective implementation of the reference architecture should coordinate all the above five key elements to meet DR Service Level Agreements (SLAs). DR SLAs typically include two metrics: Recovery Time Objective (RTO) and Recovery Point Objective (RPO). RTO specifies the time period within which recovery has to be completed, while RPO bounds the amount of data loss permitted during the recovery.

Failover refers to the process of recovery upon declaration of disaster. Disaster declaration may actually precede occurrence of a real disaster, e.g., in anticipation of extreme weather conditions. The failover or recovery procedure typically consists of three phases: environment recovery, workload recovery, and management recovery. The environment recovery phase recovers networking and critical management components so that they are functional enough to support workload recovery. The workload recovery phase recovers customer VMs, customer applications, and cloud services that those VMs and applications require. Finally, the management recovery phase recovers capabilities of the cloud needed for managing customer workloads. We provided an overview of the three phases at the cloud infrastructure level in previous work [1]. The focus of this paper is application-level recovery in a PaaS or IaaS cloud environment.

III. Infrastructure-level versus Application-level DR Solutions

Broadly, there are two levels at which DR protection and recovery can be performed for cloud-hosted enterprise applications: infrastructure-level and application-level. Infrastructure-level DR protection and recovery is typically done at the storage-level or (virtual or physical) server-level. Storage-level protection is based on replication that works by copying any updates to servers' state at the primary site's storage to the secondary site's storage. Such replication relies on mirroring features of the underlying storage subsystem (available in certain enterprise-class storage systems, e.g., SAN Volume Controller [19]). An advantage of storage-level replication is that it is agent-less and it can cover all servers (physical or virtual) and all clients. Server-level protection uses server-level replication, which does not require special features of storage system, but requires installation of a replication agent in each (physical or virtual) server. The per-server agent installation may be cumbersome and different versions of the agent on different operating systems may need to be developed. Application-level protection and recovery is needed for certain services, applications, or middleware because generic storage-level or VM-level replication may not satisfy their special requirements. For example, if SAP HANA [21] appliances are replicated using a non-SAP-certified mechanism, SAP support and warranty become void. So, we have to use *HANA System Replication*, the SAP-certified replication mechanism for HANA appliances. Application-level protection and recovery mechanisms are usually specific to the particular service, application, or middleware they are intended for, and may not be re-usable across other services, applications or middleware. In the rest of the paper, we use the term "application-specific DR" to refer application-level protection and recovery solutions.

Infrastructure-level DR treats a cloud application as a set of processes hosted on multiple machines at the primary cloud site. Failover is performed by running equivalent machines on the DR site, restoring them with data replicated prior to disaster occurrence, and executing the same processes whose state is (re-)initialized from the replicated data. Thus, infrastructure-level DR is concerned with recovering the entire state of all machines hosting an application. In contrast, replication and state recovery in application-level DR solutions are concerned only with the state of the application itself. Hence, application-specific DR solutions are more efficient in DR-protecting an application than infrastructure-level solutions. However, from a failover management and orchestration perspective, when a large number of applications spanning many cloud users need DR protection, it may be more convenient and robust to use infrastructure-level DR instead of application-specific DR.

Consider a cloud application that needs to be DR-protected. Even if infrastructure-level cloud DR solutions are available, it may not be applicable for DR-protecting the application in the following circumstances:

1. Contractual or licensing terms of the application or underlying middleware stipulate the use of vendor-provided replication and recovery mechanisms (e.g., SAP HANA).
2. Infrastructure-level replication is performed at the storage layer, but the replication is not compatible with the storage devices and appliances used by the application.
3. Infrastructure-level replication may not meet the performance requirements of the application.
4. Application-level or underlying middleware-level consistency cannot be guaranteed by infrastructure-level DR solution.
5. The application requires that the phases of the DR lifecycle (e.g., steady-state replication, failover, and failback [1]) be executed at a finer granularity than allowed at the infrastructure-level. For example, it may be necessary to quiesce collections of application components spanning several VMs prior to taking a snapshot or performing replication to the DR site.

6. Certain applications (e.g., SAP HANA [2][21]) can be deployed on geographically distributed nodes, have in-built state-sharing schemes that are efficient over long-distances, and are amenable to flexible routing of production traffic to separate instances of the application. Such applications are natural candidates for DR protection using application-specific replication and recovery.

IV. Infrastructure-level DR Solutions

We now describe infrastructure-level DR protection for a cloud application. Two components are necessary for workload protection: data replication and recovery.

A. Data Replication

A number of infrastructure-level data replication techniques are available for DR of cloud applications [9-17]. Infrastructure-level DR solutions typically use storage-based replication or host-based replication. In storage-based replication, the cloud storage system replicates the storage volumes to the storage volumes of the secondary site (e.g., [19]). In a host-based replication scheme, an agent is installed on the host (physical machine or virtual machine), and the agent intercepts all writes to storage and transmits these writes to the secondary site. The important notion of *consistency group* refers to write-order consistency (i.e., a total ordering of writes) across multiple disks belonging to one or more VMs and is supported to varying degrees by different replication methods.

In *active-active* replication, secondary VMs are provisioned at the secondary site as soon as the primary VMs are chosen for DR protection (i.e., before the actual occurrence of a disaster). Data can then be replicated directly into the storage of the secondary VMs. In *active-passive* replication, replicated data are placed in storage of the secondary site, but VMs are not provisioned until a disaster occurs and recovery is initiated. Active-active replication is costlier than active-passive replication because resources are consumed by the recovery VMs even when there is no disaster, but it achieves shorter RTO because there is no need to provision VMs during failover.

B. Recovery

From the infrastructure point of view, a service or an application comprises of a number of (physical or virtual) machines. The machines and the workloads on them are recovered at the secondary site upon a disaster. If active-passive replication is used, machines first have to be provisioned at the secondary site during the recovery. The capacity and speed at which resources can be provisioned at the secondary site may limit how many machines belonging to an application may be recovered concurrently.

1) Scheduling of Machine Recovery

Machines hosting a cloud application or service typically have complex logical dependencies among each other. These inter-dependencies must be observed when recovering the machines during failover. For example, when recovering a three-tier web application, the database server is started before the application server. That allows the application server to connect to the database server prior to the launch of applications hosted on the application server. Automated logical dependency discovery techniques via configuration, traffic, and code analysis are well-known and implemented in commercial and research systems (e.g., [26][27]). Such techniques can be used to discover dependency information between applications, middleware, data, and the machines they reside on.

Different applications in an enterprise have different values to the business, have different impacts to the business if rendered unavailable, and hence have different RTOs. Business Impact Analysis (BIA) [28] is often used to evaluate the potential impact of interruption to different parts of the business and to establish recover priorities should there be an outage. The machines hosting business applications inherit their recovery priorities from those applications.

For proper recovery planning, it is important to understand the recovery time for each constituent machine hosting a distributed cloud application. A number of factors should be considered when estimating the recovery time of each constituent machine such as the resource requirements of the secondary machine, the provisioning times and resource allocation times at the secondary site for the machine, the boot time for the operating system, and the launch times for the application processes.

A complete DR solution for a cloud service or application should include mechanisms for specifying or discovering the inter-dependencies among the involved machines and the recovery priority values of the machines, and for estimating the recovery times of the machines. These mechanisms are represented by the *Discovery and Plan* components of the DR reference architecture in Figure 1.

2) Recovery Scheduling Algorithm

Once a business function's inter-machine dependencies and recovery priorities are known, recovery must be scheduled so as to minimize the total recovery time of the business function, while complying with the dependencies and recovering high-priority machines as quickly as possible. Often, a business function's RTO can only be achieved by recovering multiple machines concurrently. For this purpose, we describe an algorithm that consumes an *acyclic* recovery graph (based on dependencies) and feeds recovery instructions to a parallelized recovery engine.

We use v to represent a node, or machine to be recovered, with a total of n nodes. We use the notion of *workers* to represent the secondary site's recovery capabilities – a given worker can recover one virtual machine at a time. We use w to represent a worker, with a total of m workers that can be concurrently active.

$P(v)$: priority of the node v set by the administrator. $P(v)$ is an external input to this algorithm, typically obtained from a Business Impact Analysis.

$T(v)$: estimated recovery time for the node v alone.

$FP(v)$: adjusted final priority value of the node v.

$ET(v)$: aggregate estimated recovery time for the node v and all nodes that, directly or indirectly, depend on v. $ET(v)$ counts in the estimated recovery time of all nodes that depend on v because, in general, a node that has more nodes depending on it with a larger waiting time should be recovered earlier.

$AT(v)$: aggregate actual recovery time for the node v, starting from the very beginning of application recovery.

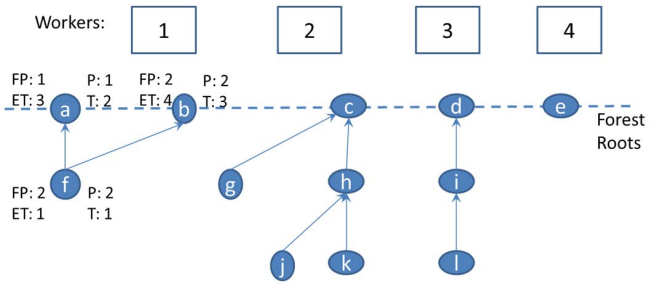

Figure 2: Forest of Nodes for Recovery Scheduling

The recovery scheduling algorithm selects nodes to be recovered by utilizing any idle workers, so as to minimize Max(*AT(v)* for all nodes *v*), while complying with the dependency relationships. In particular, the algorithm will not select a node *v* for recovery until all nodes that *v* depends on have been recovered.

The formulated problem is a variant of the job-shop scheduling problem [29] with following distinguishing features: (i) the actual recovery time *AT(v)* is not a fixed value and can differ from the estimated recovery time *ET(v),* (ii) the number of workers is dynamically changing as more or fewer resources become available for the recovery, and (iii) besides dependencies previously discussed in certain job-shop scheduling problems (in prior-art), there are also machine-associated priority values to be handled. It is known that the job-shop scheduling problem is NP-complete for $m > 2$. Solution algorithms exist for m = 2 (such as [29]). Our heuristic algorithm is specialized for recovery scheduling and planning, and applies for any number of workers *m*.

Figure 2 illustrates the data structure used in our algorithm. Nodes are organized into a forest. An arrow in the figure indicates a dependency relationship: e.g., the node *g* depends on the node *c*. Every node *v* is associated with *P(v)*, *T(v)*, *FP(v)* and *ET(v)* values. For illustration purposes, only 3 nodes are marked with sample values in Figure 2. The dash line links all nodes that do not depend on any other nodes. These nodes form the roots of the forest. Four workers are shown in the figure as an example (i.e., $m = 4$).

Heuristic Algorithm: We first construct the forest of the nodes as illustrated in Figure 2 with the *P(v)* and *T(v)* values specified. Then we recursively calculate the *FP(v)* and *ET(v)* values for all nodes. In Figure 2, we do the following calculation starting with the leaf nodes (i.e. the bottom nodes *f, g, j, k, l* in Figure 2). Note that a machine *v* with a higher priority has a lower *P(v)* value:

$$FP(v) = \begin{cases} P(v), \text{ if } v \text{ is a leaf} \\ Min(P(v),\ FP(q) \text{ for all nodes } q \text{ that directly depend on } v) \end{cases}$$

$$ET(v) = \begin{cases} T(v) + Max(ET(q) \text{ for all nodes } q \text{ that directly depend on } v), \text{ if } m \geq n \\ T(v) + \sum(ET(q) \text{ for all nodes } q \text{ that directly depend on } v), \text{ if m=1} \\ T(v) + r \cdot Max(ET(q)) + (1-r) \cdot \sum(ET(q)), \text{ for } 1<m<n \end{cases}$$

The formulas above show that the *FP(v)* should reflect the highest priority of *v* itself and all nodes that directly or indirectly depend on *v*, and the *ET(v)* calculation is dependent on the number of workers, *m*. If $m \geq n$, then *ET(v)* should be *T(v)* plus the max of the *ET* values among all the nodes that directly depend on node *v*. That is because, all

```
Algorithm select_nodes (forest_roots, num_workers)
   forest_roots: the list of nodes in the forest roots
   num_workers: how many workers available to be fed

   sort forest_roots in increasing order of their FP values (for
nodes with equal FP values, in decreasing order of their ET
values)
   initialize selected as an empty list;
   for (i=0; i<forest_roots.size(); i++) {
      if (selected.size() >= num_workers) // no idle worker
         break;
      forest_node = forest_roots.get(i);
      if (forest_node is already under recovery)
         continue;
      selected.add(forest_node);   // select forest_node
      mark forest_node as "under recovery";
   }
   return selected;
```

```
Algorithm nodes_complete (forest_roots, completed_nodes)
   forest_roots: the list of nodes in the forest roots
   completed_nodes: the list of nodes that just completed
recovery

   for (i=0; i<completed_nodes.size(); i++) {
      forest_node = completed_nodes.get(i);
      mark forest_node  as "recovered";
      forest_roots.remove(forest_node);
      next_to_recover = set of nodes {v}, such that v depends on
forest_node, and all the nodes that v depends on are "recovered";
      forest_roots.add(next_to_recover);
   }
```

Figure 3: *select_nodes* and *nodes_complete* Algorithms

those nodes with dependencies on *v* can be recovered concurrently. If *m* is 1, there is no concurrency and all machines are recovered sequentially, so *ET(v)* is *T(v)* plus the sum of the *ET* values of all the nodes that directly depend on node *v*. For any $1 < m < n$ value, we can use a heuristic ratio *r* ($0 < r < 1$) to reflect the general concurrency in the nodes to be recovered. For a highly concurrent system, we tend to select a large *r* value. A good approximation for *r* when $1 < m < n$ is *m/n*.

After calculating the *FP(v)* and *ET(v)* values for all nodes in the forest, we invoke the *select_nodes* algorithm to select nodes from the root nodes of the forest and feed them to workers. When recovery of one or more nodes is completed, we invoke the *nodes_complete* algorithm to remove the completed nodes from the root nodes of the forest, and to add the next set of nodes to be recovered to the root nodes of the forest. The *select_nodes* algorithm is also invoked whenever there are one or more workers available, e.g. when new resources are added to the secondary site. Figure 3 shows the pseudo code of the *select_nodes* and *nodes_complete* algorithms. The *select_nodes* algorithm selects nodes with lower *FP* values and lines them up for recovery by worker nodes; for nodes with equal *FP* values, the algorithm selects nodes with larger *ET* values from the forest roots.

Computation Complexity: The heuristic algorithm consumes an acyclic dependency graph and executes the recovery with maximal concurrency. The algorithm consists of four steps:

(1) constructing the initial forest (e.g., Figure 2), including computation of the *FP(v)* and *ET(v)* values of all nodes

(2) sorting the root nodes of the forest in *select_nodes()*

(3) selecting the next set of nodes to be recovered in order of priority in *select_nodes()*

(4) handling completion of node recovery in *nodes_complete()*.

Let *d* denote the maximum degree of the nodes in the dependency graph, and *k* denote the maximum number of root nodes in the forest. Then the computational complexity for the four steps are as follows:

O(n.d) for Step (1): Constructing the forest data structure requires creating all the nodes and edges; each node has at most *d* edges. When calculating *FP(v)* or *ET(v)*, all nodes *q* directly depending on *v* are visited, which results in complexity of *O(n.d)*.

O(k.logk)+O(n.k) for Step (2): The initial sort of the root nodes of the forest in *select_nodes()* takes O(*k.logk*) time. Subsequently, when recovered nodes are removed from the roots and new nodes are added into the roots, each new node is inserted into a sorted list properly so that the result list is still sorted. This takes *O(k)* for each node, and the time overhead for processing all nodes is *O(n.k)*.

O(n) for Step (3): Because the nodes are selected for recovery from the head of the sorted list of root nodes, and each selected node is visited only once.

$O(n.d^2)$ for Step (4): Upon the completion of recovery of any node, the next set of root nodes is determined. Suppose *v* is a node that depends on the recovered node. Then, Step (4) examines whether all nodes that *v* depends on are recovered. If *v* depends on *f* nodes directly, this check will be performed at most $O(f^2)$ times, with *f* no more than *d*.

Considering all the four steps above, the overall complexity is $O(max(n^2, n.d^2))$, as *k* is at most *n*. In practice, *d* is much smaller than *n*, since the scope of recovery is a cloud site consisting of many nodes and multiple applications.

V. Application-Specific DR Solutions

In Section III, we presented scenarios in which application-specific DR solutions are required or preferred for a cloud application, even if infrastructure-level DR is available. Application-specific DR solutions leverage the information and knowledge about the application that is generally unavailable at the infrastructure level. We now present a categorization of these solutions (Table 1) and two case studies based on our practical experiences in activating a subset of the solutions on the IBM CMS cloud platform [4].

A. Categorization of Application-Specific DR Solutions

A number of middleware solutions come with built-in vendor provided DR capabilities. For example, if a database service on a PaaS cloud is built on top of Oracle, then the DR solution for the database service can leverage Oracle Data Guard (ODG) [20], which is a replication and recovery mechanism for Oracle Database. Similarly, DR solutions for database services based on SAP HANA [2] can use HANA System Replication [21], while those based on IBM DB2 can use DB2 High Availability and Disaster Recovery (HADR). Sometimes, the choice of the DR solution may be restricted to the one provided by the vendor due to the need to keep the system fully supported or compliant.

Owners of applications may provide application-specific DR solutions. Cloud-native applications such as those built on top of cloud services such as AWS S3 [10], can be implemented to take advantage of the native DR features of such services and place multiple component instances in geographically distributed regions (e.g. Amazon availability zones) [10]. In addition, a service owner will have in-depth knowledge of the service, and can selectively replicate certain files while excluding other files in the relevant machines from being replicated (e.g., using *rsync* [18]).

Table 1: Categorization of Application-Specific Solutions

Provided By	Examples	Comments
Software Vendor	Oracle Data Guard (ODG) [20], Cassandra replication, SAP HANA replication [21], IBM DB2 HADR, and WebSphere Extreme Scale Distributed Web Caching.	ODG can be symmetric (standalone-standalone, cluster-cluster) or asymmetric (cluster-tandalone). Some solutions use the warm DR pattern, while others use the active-active pattern.
Service or Application Owner	DR for Amazon workloads on top of AWS services [10], rsync [18] or other file system synchronization [22][23], and custom checkpoint implementations.	The category also includes DR solutions from third-party DR-as-a-Service providers on top of AWS services.
Cloud Provider	Application-specific DR for IBM CMS dedicated appliances is integrated with infrastructure-level SAN-based Global Mirroring [19]	In addition to providing infrastructure-level DR solutions, cloud providers may play the role of integrator for certain application-specific DR solutions.

Cloud providers may also offer application-specific DR solutions that complement infrastructure-level DR solutions. For example, the IBM CMS platform uses Global Mirroring [19] based data replication for the infrastructure-level DR solution. Additionally, for some services like Database-as-a-Service, CMS integrates and orchestrates the application-level DR solution with the Global Mirroring DR solution.

Traditional workloads and services (e.g. three-tier web applications, transaction-based applications and their components) typically apply the *warm DR* pattern for their DR solutions. Until a DR declaration is made, client requests and inputs in the *warm DR* pattern are processed only by the components at the primary cloud site. Replication and control traffic are transmitted from the primary to the secondary site.

Cloud-native workloads and services (e.g. applications built on top of Amazon S3 [24], BlueMix [5], and CloudFoundry [25]) typically apply the active-active or live-live pattern for their DR solutions. Until a DR declaration is made, client requests and inputs in the active-active pattern are distributed to active application components at both primary and secondary cloud sites using mechanisms such as global load balancer and DNS re-routing.

B. Case Studies

IBM Cloud Managed Services (CMS) [4] is a cloud offering that provides a mix of virtualized and non-virtualized cloud environments for enterprise workloads. CMS offers managed services both at the IaaS level and at the PaaS level, specifically for SAP [31] and Oracle [32] applications. We describe our implementations of application-specific DR in the CMS platform for those applications.

Infrastructure-level DR solution in CMS is based on Global Mirroring. Infrastructure-level DR SLAs (i.e., virtual machine RTOs and RPOs) are the responsibility of the cloud provider (i.e., IBM). The responsibility of the cloud provider is typically limited to providing sufficient bandwidth needed to support replication for specified applications. Appropriate capacity planning is required to estimate the bandwidth and latency requirements that must be met.

Certain situations mandate the use of application-level recovery. In this mode, geographically distributable applications such as Oracle DataGuard and DB2 HADR are used for providing application-level data replication and recovery. The application owner is responsible for the application DR plan and execution, and ultimately for meeting application-level RTOs and RPOs.

In the CMS implementation, applications initiate DR processing upon receipt of a "shoulder tap" from a CMS management component called Disaster Recovery Orchestrator (DRO). The DRO is an instantiation of the *Control and Automation* component in the Reference Architecture (Section II). The DRO orchestrates the infrastructure-level DR recovery procedure, applies the shoulder tap to the secondary instance of the application within 15 minutes of initiation of DR. Thereafter, the RTO of the service/application specific DR is the responsibility of the owner of the specific DR solution. The CMS platform supports managed application offerings for SAP and Oracle. For applications covered under these offerings, IBM CMS is considered the owner of the DR solution, and will therefore be responsible for the RTO and RPO of those applications.

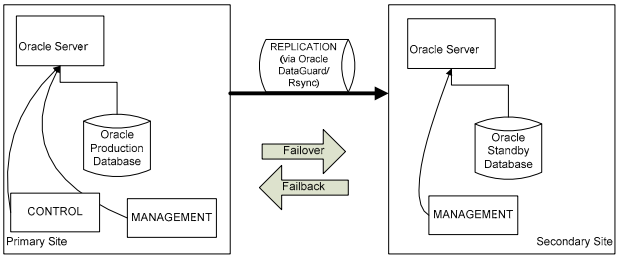

Figure 4: Standalone-to-Standalone DR for Oracle

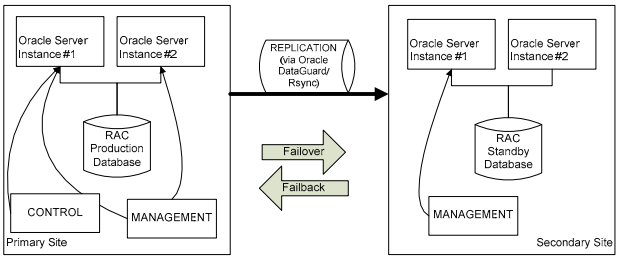

Figure 5: Cluster-to-Cluster DR for Oracle Workload

1) DR of Database Service using Oracle Database (DB)

Data replication for Oracle DB can be done using Oracle Data Guard [20] or Rsync [18]. In the CMS Database service, Oracle enterprise workload has three use cases (Figure 4, Figure 5, and Figure 6) that are divided in two topology types: *symmetric* and *asymmetric*.

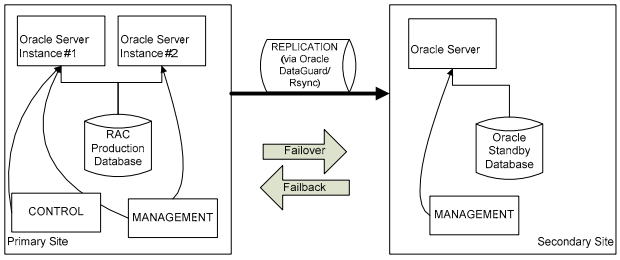

Figure 6: Cluster-to-Standalone DR for Oracle Workload

The symmetric topology requires the same server configuration at both primary and secondary sites. There are two models of symmetric topology: *Standalone-to-standalone* and *cluster-to-cluster*. Both implementations are examples of the *Warm DR* model and have similar license and other costs at the primary and the secondary sites. Standalone-to-standalone (Figure 4) is the basic DR setup for a single Oracle database and application. Figure 4 depicts a *warm DR* scenario with the secondary database on standby, waiting to be switched to active upon failover. An Oracle Real Application Cluster (RAC) cluster is often used to provide high availability on the primary site. In the cluster-to-cluster model (Figure 5), Oracle RAC replicates to both cluster nodes at the secondary site via Oracle DataGuard.

Oracle uses Oracle Data Guard (ODG) to replicate a primary database to a standby database, which runs on a server provisioned in a secondary site. However, we find ODG alone does not suffice for the database service in CMS, since replication of certain non-database file systems and directories. That was accomplished using *rsync*.

The asymmetric topology allows a cluster on the primary site to be replicated to a standalone server at the secondary site. As shown in Figure 6, RAC cluster is set up to replicate to a standalone database. This is done to reduce the licensing and other costs of the application at the secondary site.

2) DR for HANA Enterprise Workload

The HANA workload requires application-level replication (i.e. HANA System Replication, the SAP-certified replication) for keeping the HANA vendor's support and warranty intact [21]. A HANA appliance requires 5-8 Gbps network bandwidth between the primary and secondary sites. Sufficient network bandwidth must be allocated to support HANA System Replication in the DR solution.

The CMS DR solution provides DR as a fully managed service, and hence, it is crucial that the HANA instances at the primary and secondary sites must be monitored. HANA replication must also be continuously monitored and incidents in the ticketing system should be created to inform the support team for a timely response.

HANA System Replication handles the relations that exist between HANA and other SAP applications in the SAP Primary production environment, e.g., ERP (Enterprise Resource Planning) Central Component (ECC), Customer Relationship Management (CRM), and Business Information Warehouse (BW) [3]. Our DR solution leverages this relation knowledge for restoring those connections during recovery, via the scheduling algorithm described above.

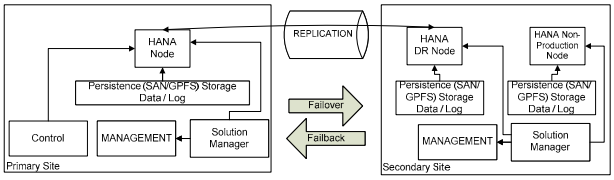

Figure 7: HANA Standalone-to-Standalone DR Topology

The two topologies for DR setup of HANA workloads are: standalone-to-standalone (Figure 7) and cluster-to-cluster. The *Solution Manager* provides management of SAP Applications. The *Control* component provides DR orchestration. The cluster-to-cluster mode is similar to Figure 7 except that the cluster scenario has multiple HANA nodes on the primary, and the secondary setup requires the same number of HANA nodes as the primary site.

a) Dual-purposing of Resources for Reducing DR Cost

In order to reduce DR costs, we can employ a hybrid of the active-passive and the active-active DR model. In this hybrid model, the HANA server at the secondary site is used to support non-production workloads (e.g., a quality assurance system) while the HANA server at the primary site is running normally. The HANA server size (e.g., CPU and memory) at the secondary site may be smaller than that at the primary site. To support dual-purpose use of the secondary server, usually 10% of system resources is required for system replication and disk expansion is required. Such changes might impact RTO as no data can be preloaded in the RAM; consequently, preload of tables must be switched off on the secondary. Once the failover from primary to secondary takes place during the disaster, non-production workloads will be shut down.

b) Synchronous HANA Replication

We can apply synchronous mode of HANA replication when the primary and secondary sites are located within 100 kilometers of each other [21]. In this mode, the SAP HANA System replication component replicates all actions performed by application processes on the primary to their counterparts at secondary site. The HANA server at the secondary site executes the exact same instructions as the primary except for accepting user requests or queries. In case of the dual-purpose server, the non-production instance of the HANA runs in parallel to the DR HANA instance, which performs all instructions of a non-production workload.

No additional license is required, since the primary replicates relevant license information to the secondary, and they both share the same System Identifier (SID). Upon start of the secondary site HANA cluster, a snapshot is taken, and each process establishes a connection to its primary site counterpart and requests the data located in main memory. Once the snapshot has been transferred, the primary site system continuously sends the log information to the secondary site system running in recovery mode. The secondary site system acknowledges and persists the logs. Thus, a new incremental data snapshot is replicated from time to time. Such a *delta* snapshot is useful for restoration in case of network/connectivity failure.

c) Asynchronous HANA Replication

When the primary and the secondary DR site are continents apart, we use asynchronous HANA SAP System Replication, in which the primary commits transaction as soon as the replicated log is sent out to the secondary. No acknowledgement is required from the secondary.

d) HANA System Replication Failover

Using the formalism for the recovery scheduling algorithm described above, we model HANA System Replication Failover and the failover of SAP applications that depend on the HANA DB (Figure 8). For a single instance of a HANA application, a recovery worker first recovers the HANA DB (a), and then N recovery workers can recover the applications (b-e) that utilize the HANA DB. The applications could be on the same or different VMs.

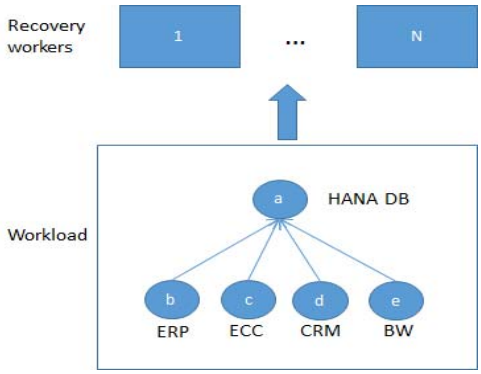

Figure 8: Recovery Scheduling – HANA System Replication

e) HANA System Replication Failover Performance Results

For the purposes of demonstrating DR performance, a Symmetric Asynchronous HANA DB DR topology was set up between Dallas and London. Replication of a 42GB HANA database was initialized and failover was manually invoked. The total time to recover the HANA DB was two minutes and two seconds, which meets the current RTO SLA of the IBM CMS offering [33]. Note that the recovery time does not include the time required to restart the applications that are using the HANA DB.

VI. Conclusions

Licensing and (occasionally) the need for specialized hardware care drive up the cost involved in setting up DR for an enterprise application. To reduce the cost of a DR solution, it may be necessary to explore downsizing specific secondary application elements without compromising on overall SLAs and functionality. Further cost reduction can be obtained by running non-production workloads on the secondary resources allocated for DR in steady-state prior to disaster declaration.

For applications that are memory bound, it is necessary to monitor them closely for memory utilization to ensure that the overhead of replication does not cause performance degradation of the application running on the primary server. For application replication, it is crucial to

ensure that the replication and application parameters (such as buffer sizes and throttling rates) are configured per the relevant specifications on both primary and secondary servers. Network delays can lead to replication failures and performance bottlenecks on the primary server.

Application-specific DR solutions can offer more efficient and finer-granularity configuration, data replication, and DR life cycle operations than infrastructure-level DR solutions. RTO and RPO metrics can be improved with highly tuned replication and recovery processes. Application-specific DR solutions often levy minimal requirements on the underlying multi-site infrastructure and can thus be infrastructure-agnostic to a large extent. On the other hand, they require sophisticated skills on part of the programmers and operators of such a solution. Determining whether such a solution is needed must take into account business requirements, licensing requirements, infrastructure capabilities, skills availability, and performance. Finally, complex enterprise workloads always consist of many different applications, only some of which are amenable to application-specific DR. Therefore, a combination of infrastructure-level and application-specific DR solutions is often needed to meet requirements in a cost-effective way.

REFERENCES

[1] Long Wang, Harigovind V. Ramasamy, Richard E. Harper, Mahesh Viswanathan, and Edmond Plattier. "Experiences with Building Disaster Recovery for Enterprise-Class Clouds." In Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on, pp. 231-238. IEEE, 2015.

[2] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees, "The SAP HANA Database--An Architecture Overview," in IEEE Data Eng. Bull. vol. 35, no. 1, pp. 28-33, 2012.

[3] L. Hossain, J. D. Patrick, and M. A. Rashid, Enterprise Resource Planning: Global Opportunities and Challenges. Hershey Park, PA: Idea Group Publishing. 2001.

[4] IBM Corporation, IBM Cloud Managed Services. [Online]. Available: http://www.ibm.com/marketplace/cloud/managed-cloud/us/en-us (Accessed: May 3, 2016)

[5] IBM Corporation, IBM Bluemix, [Online]. Available: http://www.ibm.com/Bluemix (Accessed: May 3, 2016)

[6] Google Inc., Google App Engine. [Online]. Available: https://cloud.google.com/appengine/ (Accessed: May 3, 2016)

[7] Microsoft Inc., Microsoft Azure SQL Database. [Online]. Available: https://azure.microsoft.com/en-us/services/sql-database/ (Accessed: May 3, 2016)

[8] Salesforce.com Inc., Heroku. [Online]. Available: https://www.heroku.com (Accessed: May 3, 2016)

[9] Microsoft Azure, "Site Recovery – Cloud Disaster Recovery," 2014. [Online]. Available: http://azure.microsoft.com/en-us/services/site-recovery/ (Accessed: May 3, 2016).

[10] Amazon Web Services, "Using Amazon Web Services for Disaster Recovery," 2014. [Online]. Available: https://media.amazonwebservices.com/AWS_Disaster_Recovery.pdf (Accessed: May 3, 2016).

[11] VMware, "VMware vCloud Air Disaster Recovery," 2015. [Online]. Available: http://www.vmware.com/files/pdf/vchs/VMware-vCloud-Hybrid-Service-Disaster-Recovery-DS.pdf (Accessed: May 3, 2016).

[12] Evault, "Cloud Disaster Recovery Services," 2015. [Online]. Available: http://www.evault.com/products/disaster-recovery/ (Accessed: May, 2016).

[13] IBM Global Technology Services, "Virtualizing disaster recovery using cloud computing," 2015. [Online]. Available: https://www-935.ibm.com/services/uk/en/it-services/vsr_whitepaper.pdf (Accessed: May 3, 2016).

[14] Bill Robbins, "Disaster recovery in a cloud environment," 2011. [Online]. Available: http://www.ibm.com/developerworks/cloud/library/cl-disasterrecovery/cl-disasterrecovery-pdf.pdf (Accessed: May 3, 2016).

[15] Rackspace, "IT High Availability Disaster Recovery Plans with Rackspace," 2015. [Online]. Available: http://www.rackspace.com/disaster-recovery-planning (Accessed: May 3, 2016).

[16] RightScale, "Disaster Recovery." [Online]. Available: http://www.rightscale.com/solutions/cloud-computing-uses/disaster-recovery (Accessed: May 3, 2016).

[17] CSC, "Business Continuity Management and Disaster Recovery," 2015. [Online]. Available: http://www.csc.com/cybersecurity/offerings/45397/86285-business_continuity_disaster_recovery (Accessed: May 3, 2016).

[18] Rsync, "Rsync." [Online]. Available: https://rsync.samba.org/ (Accessed: May 3, 2016).

[19] IBM, "Global Mirror Technical Whitepaper," 2008. [Online]. Available: http://www-01.ibm.com/support/docview.wss?uid=tss1wp100642 (Accessed: May 3, 2016).

[20] Oracle, "Data Guard Concepts and Administration," 2008. [Online]. Available: http://docs.oracle.com/cd/B19306_01/server.102/b14239/toc.htm (Accessed: May 3, 2016).

[21] SAP, "SAP HANA." [Online]. Available: http://hana.sap.com/ (Accessed: May 3, 2016).

[22] Aspera Synch, "Aspera Synch," [Online]. Available: http://asperasoft.com/software/synchronization (Accessed: May 3, 2016).

[23] R. H. Patterson, S. Manley, M. Federwisch, D. Hitz, S. Kleiman, and S. Owara, "Snapmirror: File-system-based asynchronous mirroring for disaster recovery," In Proceedings of the 1st USENIX Conference on File and Storage Technologies, FAST 2000.

[24] Amazon Web Services, "AWS Simple Storage Service (S3)." [Online]. Available: https://aws.amazon.com/s3/ (Accessed: May 3, 2016).

[25] CloudFoundry, "CloudFoundry." [Online]. Available: https://www.cloudfoundry.org/ (Accessed: May 3, 2016).

[26] IBM Corporation, "Analytics for Logical Dependency Mapping (ALDM)." [Online]. Available: http://www-935.ibm.com/services/us/en/it-services/data-center/it-infrastructure-discovery/ (Accessed: May 3, 2016).

[27] K. Magoutis, M. Devarakonda, N. Joukov, and N. Vogl. "Galapagos: Model-driven discovery of end-to-end application storage relationships in distributed systems," IBM J. Research, 52:367–378, 2008.

[28] M. Swanson, P. Bowen, A. W. Phillips, D. Gallup, and D. Lynes. Contingency Planning Guide for Federal Information Systems. NIST Publication No. 800-34. Rev. 1, NIST, Nov 2010.

[29] M.R. Garey, "The Complexity of Flowshop and Jobshop Scheduling," Mathematics of Operations Research 1 (2): 117–129, 1976.

[30] Yousef Khalidi, "Building a Cloud Computing Platform for New Possibilities", *Computer*, vol.44, no. 3, pp. 29-34, March 2011.

[31] IBM Corporation, IBM Cloud Managed Services for SAP Applications. [Online]. Available: http://www-935.ibm.com/services/us/en/it-services/cloud-services/cloud-managed-services-SAP/index.html (Accessed: May 3, 2016)

[32] IBM Corporation, IBM Cloud Managed Services for Oracle Applications. [Online]. Available: https://www.ibm.com/marketplace/cloud/cloud-services-for-oracle-applications/us/en-us (Accessed: May 3, 2016)

[33] IBM Corporation, IBM Cloud Managed Services Disaster Recovery. [Online]. Available: http://www-935.ibm.com/services/us/en/it-services/cloud-services/cloud-managed-services-disaster-recovery (Accessed: May 3, 2016)

[34] Peter M. Mell and Timothy Grance, "The NIST Definition of Cloud Computing," NIST Technical Report SP 800-145, Gaithersburg, MD, United States.