Performance Analysis of Multi-Institutional Data Sharing in the Clouds4Coordination System

Ioan Petri[a,*], Omer F. Rana[a,**], Tom Beach[b], Yacine Rezgui[b]

[a]*Cardiff University, School of Computer Science & Informatics, Wales, United Kingdom*
[b]*School of Engineering, Cardiff University, United Kingdom*

Abstract

Cloud computing is used extensively in Architecture/ Engineering/ Construction projects for storing data and running simulations on building models (e.g. energy efficiency/environmental impact). With the emergence of *multi-Clouds* it has become possible to link such systems and create a distributed cloud environment. A multi-Cloud environment enables each organisation involved in a collaborative project to maintain its own computational infrastructure/ system (with the associated data), and not have to migrate to a single cloud environment. Such infrastructure becomes efficacious when multiple individuals and organisations work collaboratively, enabling each individual/ organisation to select a computational infrastructure that most closely matches its requirements. We describe the"Clouds-for-Coordination" system, and provide a use case to demonstrate how such a system can be used in practice. A performance analysis is carried out to demonstrate how effective such a multi-Cloud system can be, reporting "aggregated-time-to-complete" metric over a number of different scenarios.

Keywords: Cloud Computing, Cloud Federation, Performance Analysis, Distributed Coordination, CometCloud, Architecture, Engineering and Construction (AEC) Projects

*Ioan Petri
**Omer F. Rana
Email addresses: petrii@cardiff.ac.uk (Ioan Petri), ranaof@cardiff.ac.uk (Omer F. Rana), beachth@cardiff.ac.uk (Tom Beach), rezguiy@Cardiff.ac.uk (Yacine Rezgui)

1. Introduction

Cloud computing enables applications in the Architecture/ Engineering/ Construction (AEC) to dynamically scale-up (increasing volume of data/computation) & scale-out (increasing diversity of computational infrastructures involved). This becomes more relevant when projects are being undertaken by a consortia of companies, who work collaboratively for the duration of the project. Such projects are complex and the consortia members provide a range of skills to the project from its inception to completion. During this process, various data artifacts are also generated that need to be stored and shared between project members (generally using access control strategies – which limit what can be accessed at a particular stage of the AEC project lifecycle). The planning, implementation and running of these AEC industry projects requires the formation of secure Virtual Enterprises (VEs) to enable collaboration between its members by sharing project information and resources. An important feature of the consortia is that they are dynamic in nature and are formed for the lifetime of the project. Members can participate in several consortia at the same time and can join or leave a consortium as the project evolves.

Cloud computing offers an important computing infrastructure to facilitate the establishment and coordination of such VEs. As well as remote access, Cloud computing also provides enhanced security, including single sign-on capability, security between consortia members, simple setting up of networks to support VEs, distribution of computationally intensive jobs across multiple distributed processors (based on shared information about available resources). Each organisation involved in a VE may have access to its own Cloud computing system (privately managed internally within the organisation, or acquired through a public provider such as Amazon.com or Microsoft (via their Azure platform)). It is unlikely that all members of a consortium will share the same platform. Integrating capability across multiple platforms is therefore an essential requirement for such VEs to function in an efficient and reliable manner. The alternative would be for all members of the consortia to migrate to the same platform. Unless there is industry-wide (or consortia-wide) agreement on what this platform should be, such a migration process can be costly and error prone, and often the outcome of the process for a specific project can be unclear.

Various efforts have been proposed to implement such multi-Clouds, ranging from research efforts focused on Cloud interoperability e.g. the Open Cloud Computing Interface (OCCI) efforts at the Open Grid Forum [1]. OCCI provides an API and a set of protocols to enable management capability to be carried out across multiple Cloud providers. A variety of implementations are currently available, in systems such as OpenStack and OpenNebula (two open source Cloud platforms). An alternative approach to interoperability is through the development of specialist gateway nodes which enable mapping between different Cloud systems and the implementation of specialist gateways to connect different Cloud systems, the development of a Cloud Operating System (CloudOS) to connect distributed Clouds to the use of specialist in-network capability to process data in network elements between different end points (GENICloud [2]). Similarly, on-line sites such as CloudHarmony [3] report over 100+ Cloud providers that offer capability ranging from storage and computation to complete application containers that can be acquired at a price, primarily using service-based access models. As the multi-Cloud market and associated number of Cloud providers who could offer services in such a market increase in number, there is often a need to understand which Cloud providers are likely to be of most benefit in the context of a given application requirement. Matchmaking becomes an important capability in such a marketplace – enabling application users to map their requirements to infrastructure capability that may be hosted across a number of different types of Cloud systems.

This paper has two major contributions: (i) we present the implementation and use of a distributed Clouds4Coordination system used to coordinate large construction projects based on requirements of the AEC sector. We emphasise the need to aggregate capability across multiple systems, rather than require all project members to migrate to a single system, and (ii) we demonstrate the capability of the system to provide the required quality of service and functionality in use by running performance analysis and measurement. Our approach involves the implementation of a logical "shared" space that is physically distributed across multiple sites involved in a cloud federation. Such a shared coordination space enables various project members to interact with each other during the stages of a project. We compare our approach to general cloud federation efforts, specifically adapted for the needs of the AEC industry in Section 2. In Section 3 we

present the CometCloud system and how this system has been used to create the federated cloud framework, followed by a description of the "Cloud4Coordination" (C4C) system and the associated Application Programming Interface (API) that makes use of CometCloud in Sections 4 and 5. In Section 6 we evaluate the C4C system in terms of its performance with a scenario, and provide overall conclusions in Section 7.

2. Related work

Through the federation of cloud systems it has become possible to connect local infrastructure providers to a common framework where participants can exchange data and collaborate. The mechanisms used to support cloud federation can bring substantial benefits for service providers by offering facilities for accessing global services instead of increasing costs associated with building new infrastructure (which may not be fully utilized and may only be needed to support peaks in workload over short time frames). More importantly, organisations with spare capacity in the data centre are now provided with a simple way to monetize that capacity by submitting it to the marketplace for other providers to buy, creating an additional source of revenue. Even if computational infrastructure was made available, it may not be possible to host services or data due to issues associated with licensing and intellectual property. Federation in cloud systems has led to a real *democratisation* of cloud markets – enabling businesses to make use of a variety of different cloud providers in different geographic areas. A federated cloud also enables users to host applications with their cloud provider of choice – thereby making local decisions about pricing, software libraries/ systems and deployment environments, while still being able to connect to other computational resources.

In a federation context there are several parameters that need to be considered in order to determine the type of interactions possible between sites. When two or more sites come together, it is important to identify not only the incoming workload of each site but also the cost of outsourcing to resources managed externally, the revenue obtained from outsourcing tasks or the cost of maintaining a reasonable level of utilisation. Identifying a set of such parameters is a challenging task due to the variability in the parameters of a federated environment (such as number of

resources allocated to local vs. remote jobs, how many jobs to outsource to another site, the time interval over which access to remote jobs should be allowed, etc) and the fluctuation of resource demand. Depending on the value of such parameters, a site manager must decide whether to outsource resources, compute tasks locally or reject remote task requests altogether [6].

Our system shares commonalities with these federation solutions in terms of general *cloud bursting* and *cloud bridging* policies related to data sharing and task execution. Beyond these general terms also we differ in the type of coordination approach used. We are more focused on creating a collaboration environment between sites and supporting a data-aware workflow where users can access real time project information and make decisions according to project stages. We therefore use the core principles of federation to enable a more controlled environment where each user contributes based on roles and responsibilities.

3. CometCloud Federation

Through the federation of Cloud systems it has become possible to connect local infrastructure providers to a global marketplace where participants can transact (buy and sell) capacity on demand. The mechanisms used to support cloud federation can bring substantial benefits for service providers by offering facilities for accessing global services instead of increasing costs associated with building new infrastructure (which may not be fully utilized and may only be needed to support peaks in workload over short time frames). More importantly, organisations with spare capacity in the data centre are now provided with a simple way to monetize that capacity by submitting it to the marketplace for other providers to buy, creating an additional source of revenue [8].

The federation model is based on the Comet coordination "spaces" – an abstraction, based on the availability of a distributed shared memory that all users and providers can access and observe, enabling information sharing by publishing requests/offers to/for information to this shared memory. In particular, we have decided to use two kinds of spaces in the federation. First, we have a single federated management space used to create the actual federation and orchestrate the different resources. This space is used to exchange any operational messages for

discovering resources, announcing changes at a site, routing users' request to the appropriate site(s), or initiating negotiations to create ad-hoc execution spaces. On the other hand, we can have multiple shared execution spaces that are created on-demand to satisfy computing needs of the users. Execution spaces can be created in the context of a single site to provision local resources or to support a *cloudburst* (i.e. when additional capacity is needed to respond to a sudden peak in demand) to public clouds or external high performance computing systems. Moreover, they can be used to create a private sub-federation across several sites. This case can be useful when several sites have some common interest and they decide to jointly target certain types of tasks as a specialized community.

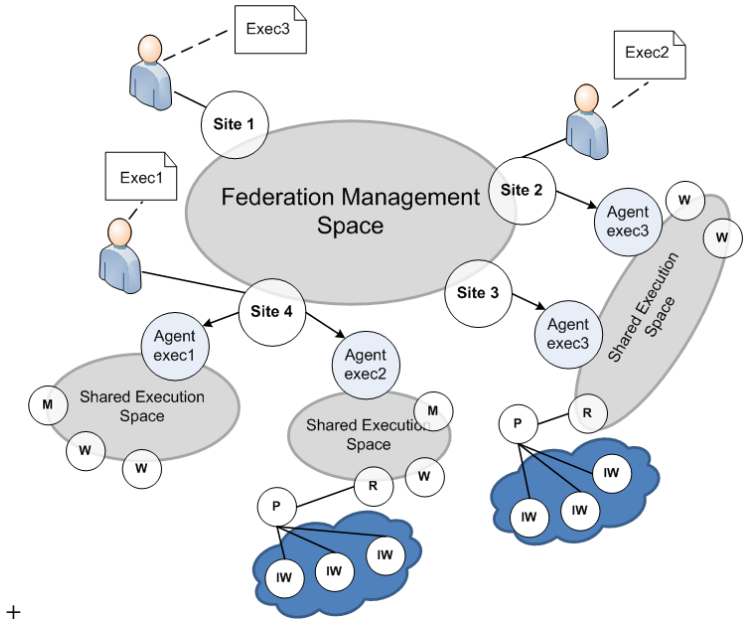


Figure 1: The overall Federation Management Space, here (M) denotes a master, (W) is a worker, (IW) an isolated worker, (P) a proxy, and (R) is a request handler.

As shown in Figure1, each shared execution space is controlled by an agent that initiates the creation of such a space and subsequently coordinates access to resources for the execution of a particular set of tasks. Agents can act as a master node within the space to manage task execution, or delegate this role to a dedicated master (M) when some specific functionality is required. Moreover, an agent deploys a number of workers to carry out execution of tasks. These

workers can be in a trusted network and be part of the shared execution space, or they can be hosted on external resources such as a public cloud and therefore in a non-trusted network. The first type of worker is called a "secure worker" (W) and can pull tasks directly from the space. Meanwhile, the second type of worker is called an "isolated worker" (IW) and cannot interact directly with the shared space. Instead, they have to interact through a proxy (P) and a request handler (R) to be able to retrieve task information from the space and execute these.

3.1. CometSpace

CometCloud uses a Linda-like tuple space [7] referred to as "CometSpace" which is implemented using a Peer-2-Peer overlay network. A tuple space enables the implementation of an associative memory-based search strategy, whereby the search term is described as a set of items/terms, which can be mapped against a table of stored data. This search strategy is often easier to implement in hardware and therefore provides a significant improvement in search performance. As an illustrative example, consider that there are a group of data producers and consumers, producers post their data as tuples in the space, and consumers then retrieve data that match a certain pattern. The producers/consumers only have a reference to where such data items should be posted/retrieved from, but do not need to know the physical location/ storage device for such data items. CometSpace is an extension to this tuple space-based abstraction, in that the tuple space can be physically distributed across multiple sites, and a "logical" space is produced by combining these physically distributed sites. Each producer/ consumer now accesses the logical space, asynchronously, and does not need to know the physical location of the site actually hosting the data [10]. In this way, a virtual shared space for storing data can be implemented by aggregating the capability of a number of distributed storage and compute resources. CometCloud therefore provides a scalable backend deployment platform that can combine resources across a number of different cloud providers dynamically, often seen as a key requirement for a project in the AEC sector.

CometCloud is based on a decentralized coordination substrate, and supports highly heterogeneous and dynamic cloud infrastructures, integration of public/private clouds and cloudbursts. The coordination substrate (based on a distributed Linda-based model) is also used to support

a decentralized and scalable task space that coordinates the scheduling of tasks, submitted by a dynamic set of users, onto sets of dynamically provisioned workers on available private and/or public cloud resources based on their Quality of Service (QoS) constraints such as cost or performance. These QoS constraints along with policies, performance history and the state of resources are used to determine the appropriate size and mix of the public and private clouds that should be allocated to a specific application request. Additional details about CometCloud can be found at [5].

For developing our C4C solution we extend the CometCloud system with the following features: (i) a BIM API based on the requirements of the AEC industry; (ii) a multi-cloud API by implementing new methods required for modelling the workflow associated with an AEC project; (iii) specification of the tuple-space mechanisms and the format of tuples to comply with requirements related to data processing, data sharing and data storage. In addition to these, we have also implemented an event propagation mechanism for monitoring the various operations appearing when working on IFC based construction projects, a metadata model for storing information about each individual object and a versioning system to record all the versions that an IFC object may have over time.

4. C4C project

In this section we outline the key industry-based requirements of the "Clouds-for-Coordination" (C4C) project. We subsequently describe the Cloud4Coordination system that has been implemented to address these requirements.

4.1. Project background

The C4C project aims to address data management challenges due to the increasing adoption of Building Information Modelling (BIM) in the AEC sector. In principle, BIM presents the possibility of sharing information throughout the construction and property management sectors. Alongside the issue of ownership, the rapid sharing of data also raises the question of trust in the data – more commonly recognised in the AEC industry through the use of 'Issue Status' for

physical documents (where documents are given statuses that equate to what they can be reliably used for, and therefore what the issuing party accepts responsibility and/or liability for). In the UK in particular this is driven by the government's aim to achieve fully collaborative Building Information Modelling (BIM) (with all project and asset information, documentation and data being electronic) across the AEC sector. This is an especially challenging proposition as the successful delivery of a construction project is a highly complex process; requiring collaboration between designers, suppliers and facilities managers through a range of design and construction tasks. This complexity in itself is a key motivation for the use of BIM, with anticipated financial and time savings offered by its adoption [12]. Other motivating factors for BIM adoption include: (a) project failure caused by lack of effective project team integration across supply chains [13, 14], (b) emergence of new challenging new forms of procurement i.e. Private Finance Initiative, Public-Private Partnership and the design-build-operate [15], and (c) decreasing the whole life cost of a building through the adoption of BIM in facilities management[16].

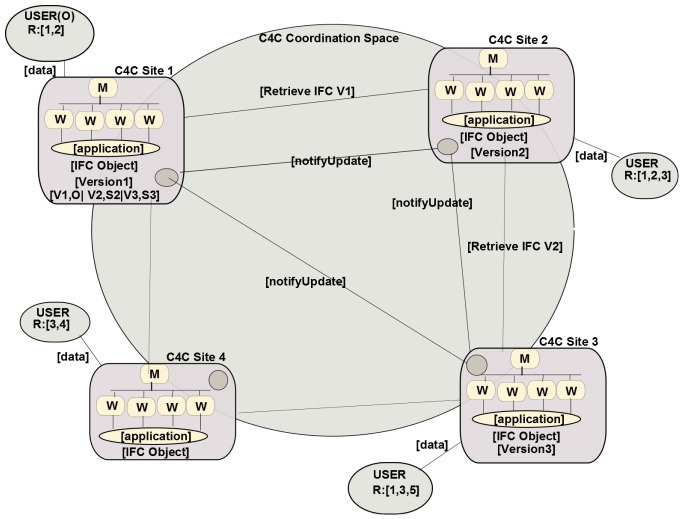


Figure 2: Clouds for coordination workflow.

The C4C project addresses the issue of BIM "ownership" by adopting the approach that each party involved creates and stores (and is responsible for) their own BIM information, rather than uploading it to a central server. More specifically our architecture imposes the following key aspects: (i) the ownership of data remains with the discipline that created that data – which also

delegates any updates needed on the data to the discipline ensuring that there is a consistent view also maintained by the discipline owner; (ii) the use of a coordination layer to allow other users to transparently view data and make modification to it; (iii) enable information to be replicated across multiple disciplines (but remain consistent with the data owner), allowing for fault tolerance and prevent data loss. Another important aspect of a management model for BIM data is understanding the data and the stages (workflow) of an AEC project, in the context of how a BIM model is populated with data. In order to do this an abstract process has been defined as the result of our requirements gathering execise. This process has abstracted the approaches defined in BS1192a[11].

Individual "nodes" that store BIM data, located at different organisations that are part of a project, must deploy agents that interact with each, using the CometCloud system. In essence, C4C will allow a complete BIM dataset to be visualised, sourced from the information stored at multiple locations (locally managed Cloud systems), without changing how or where the original source material is kept, and ensuring that the capability of the owner to revoke and manage updates is not affected. The project goal is to create a framework for AEC project information "Issue Status", which recognises both the issuing party's status (and consequentially the responsibility/liability associated), as well as acknowledging the receiving party's need or reliance on the data.

4.2. Project implementation

In the C4C project we consider that each site (organisation) involved in a particular project can have one master (agent) and several workers. We have also considered the scenario where a new site may be added during the lifetime of the project, for instance, when a project member may gain access to additional data centres. For addressing these requirements we have developed a multi-cloud API which provides all the necessary operations for managing collaboration once an AEC project has been initiated and launched.

We implement a multi-cloud API for creating publishers, subscribers and exchanging messages within our CometCloud-based system. The key benefit of the publisher-subscriber model enables us to associate a distinct discipline reference with each data producer. A user belonging

to a particular discipline (e.g. architect, electrical engineer, mechanical engineer etc) is able to have limited visibility of BIM objects across the different sites that are part of a particular project. What is visible within a specific discipline is dependent on: (i) the current stage at which particular data has been produced; (ii) the maturity of the generated object – referenced through a "suitability" level. Both of these parameters are AEC industry specific requirements, and ensure that objects can be managed and updated without conflict during the lifetime of the project. By using the publisher-subscriber model we enable sites to interact with each other on a common project, using publishers to generate project tasks and subscribers to execute these tasks. We consider the following properties for a site:

- Industry Foundation Class (IFC) objects: a generic language and data model for each of the sites in the coordination space. In our C4C framework we operate with IFC objects.
- Roles/Disciplines: we consider that sites can have different roles/ disciplines – which are considered when propagating notification messages associated with updates to particular IFC objects, i.e. which site should be involved at project collaboration stage.

Each site must support a local C4C environment, which enables other sites to interact with it. In the workflow presented in Figure 3, Site 1 creates the C4C project which is formed of IFC objects locally stored as $Version1$. All other sites participating in the project (Site 2 and Site 3) will be notified about the new project being created (based on their roles in the project). Based on the notification, Site 2 retrieves and updates the C4C project with Version 1, Site 2 then creates a new version of the C4C project as $Version2$. When a new version is created the interested sites are again notified. Site 3 will also retrieve the latest version $Version2$ and apply updates as part of a new project version – $Version3$. Another round of notifications will be propagated to interested sites (Site 1 and Site 2). Site 4, although part of the coordination space, is expected to contribute to the project at later stages thus will not receive a notification event. It is important to note that Site 1 is the owner of the project, along with the organisation that creates the project and can always retrieve the latest version of the C4C project. In addition, Site 1 also keeps a list of the changes that have been applied to the C4C project over time in a "provenance" (metadata)

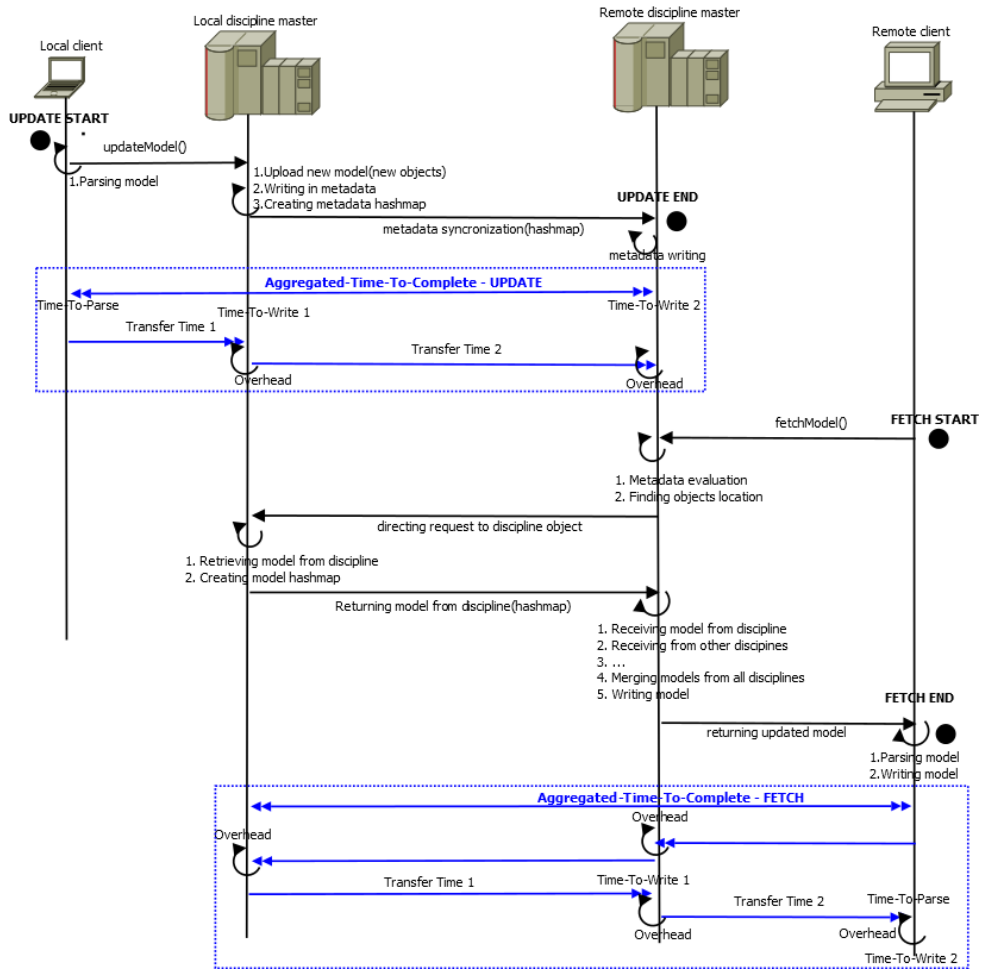


Figure 3: Aggregate-time-to-complete metric and workflow

file.

This workflow is supported by the modules that we have implemented to facilitate project coordination and IFC data sharing. In this respect, we have implemented an event propagation mechanism for monitoring the various operations than appear when working on IFC based construction projects, a metadata model for storing information about each individual object and a versioning system to record all the versions that an IFC oject may have over time. All these modules serve to the provenance model that enables users to monitor and observe the system and make informed decisions.

5. C4C Application Programming Interface (API)

In this section we present the two APIs that have been developed as part of our system: (i) an API to support multi-cloud use based on the publisher-subscriber (master-worker) model (presented in Table 1) and (ii) a BIM-based API to comply with industry standards (presented in Figure 4). For facilitating disciplines to use the background of a project we have developed methods for manipulating IFC objects and corresponding metadata. We have also developed a set of methods for enabling the distributed manipulation of these IFC objects where various disciplines associated with a project can work on the same IFC model.

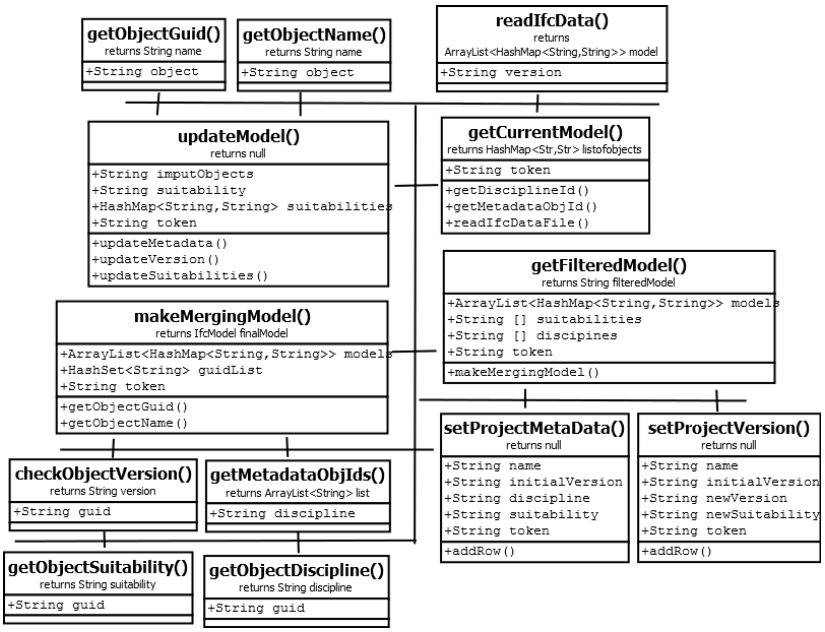


Figure 4: The C4C BIM API

The resulting functionality supports multi-cloud operation carried out over an IFC model, by providing mechanisms to transfer data between different disciplines. This allows disciplines to retrieve, in real-time, the latest version of an IFC object and to reconstruct the IFC model accordingly. Table 1 presents how the multi-cloud API can be used to enable collaboration between different partner sites.

We assume that each discipline has access to a cloud/data centre. The framework is initialized

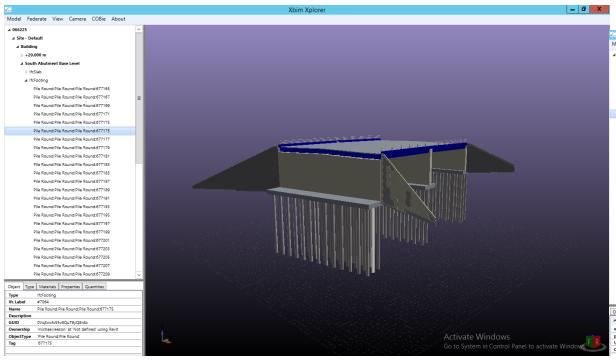
Table 1: Multi-cloud API

METHOD	DESCRIPTION
addC4CBootStrapNodes()	Sets the bootstrap node
addPorts()	Adds ports for later configurations
bootstrapnodeIsUp()	Checks for any working bootstrapnode
createC4CMaster()	Creates a new master
createC4CWorker()	Creates a new worker
createC4CMasterGeneric()	Implements a generic master
findFreePort()	Looks for available free ports
isBootstrapNode()	Compares the current node with the bootstrapNode
sendMsg()	Sends a message to a destination IP on a specific port
sendMsgToAll()	Sends local subscription list to all nodes(not to bootstrapnodes)
startC4CManager()	Starts federation by creating a master and worker
startC4CWorker()	Starts a C4C local worker
startC4CMasterServer()	Starts a local C4C master
startC4CIsolatedWorker()	Starts a C4C isolated local worker
checkAvailableC4CWorker()	Checks for one available worker
checkAvailableC4CWorkers()	Checking for all available workers based on the number of tasks
getAvailableC4CWorker()	Checks for an idle worker
createTaskData()	Creates data associated with a task
getTaskInfo()	Retrieves task info. based on *taskID*
selectC4CWorkerCreateTask()	Selects a worker, then creates a task to insert to tuple space

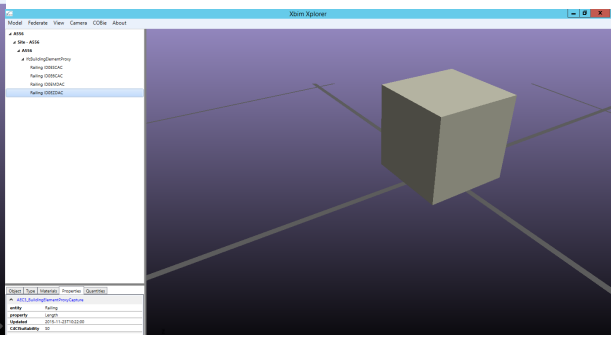
by calling "startC4CManager()" which then creates the Masters and the Workers based on specific configuration files. If a site is not set to be a Master then the C4CManager will create a proxy in order to connect with the existing data centre Worker by calling "createIsolatedWorker()" method. After the multi-cloud entities have been created, the C4CManager starts all the associated Masters and Workers by calling "startC4CMasterServer()" and "startC4CWorker()" respectively.

6. Evaluation

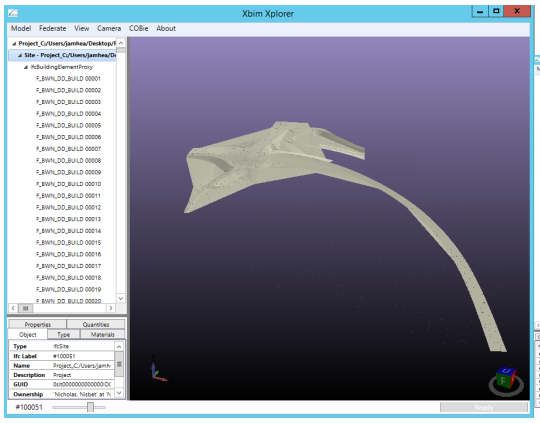
We use as input IFC models of different sizes to investigate the time taken to exchange data between sites. This preliminary set of experiments are used to benchmark the system, and investigate estimate potential management overhead. The AEC project being considered is a bridge structure with auxiliaries, which involves different disciplines contributing to various parts of the structure. We use four disciplines:(i) C-Contractor, (ii) Q-Cost Consultant, (iii) E-Engineer, (iv) O-Client. The IFC models sizes that we utilise in the experiments are: 250MB, 145MB, 3.44MB, 48KB. These input models and the output model are presented in Figure 7.



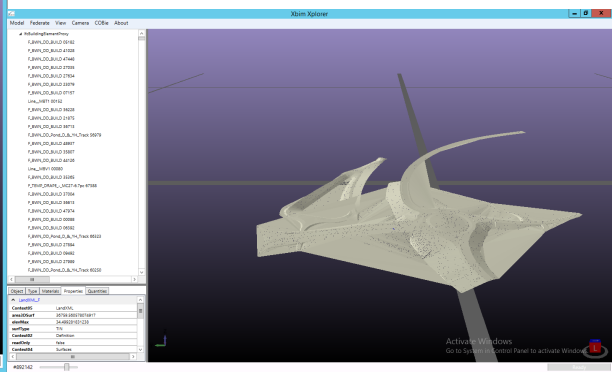
(a) Input IFC Model: Size 3.44MB



(b) Input IFC Model: Size 48KB



(c) Input IFC Model: Size 256MB



(d) Output IFC Model: Size 366MB

Figure 5: Input and output models

We use an IBM Softlayer [1] virtualized cluster-based infrastructure at the Amsterdam Data Centre, with dedicated virtual servers. To conduct the experiments we use three different system configurations:

- Configuration 1 (used in subsection 6.1): In the preliminary evaluation we deploy four Softlayer virtual machines (VMs), where each VM runs with 1 CPU core at 3.2 GHz and an IFC model of size 3.44MB (illustrated in Figure 5a). Each VM uses one core with 1GB of memory. The networking infrastructure is 1Gbps Ethernet with a latency of 14 ms on average. Each VM runs Ubuntu 12.4 and Java 7.

[1] https://control.softlayer.com/ Last accessed: Aug 2015

- Configuration 2 (used in subsection 6.2): In the second part of the evaluation we increase the server specification to 16CPU cores with 64GB of memory in Softlayer and use IFC models of size 145MB, 250MB, 3.44MB and 48KB respectively (illustrated in Figure 5). The networking infrastructure is 1Gbps Ethernet with a latency of 14 ms on average. Each server runs Ubuntu 12.4 and Java 7.

- Configuration 3 (used in subsection 6.3): In this scenario we use our local cluster system and deploy four KVM virtual machines (VMs) with 1 CPU core at 3.2 GHz each. Each VM uses one core with 1GB of memory. We use IFC models of size 145MB, 250MB, 3.44MB and 48KB respectively (illustrated in Figure 5). The networking infrastructure is 1Gbps Ethernet with a latency of 0.42 ms on average. Each server runs Ubuntu 12.4 and Java 7.

We run performance analysis experiments with the objective to identify how our system reacts when using different system configurations: (i) small model and low specs infrastructure (in Subsection 6.1 with Configuration 1), (ii) large models and high specs infrastructure (in Subsection 6.2 with Configuration 2) and (iii) large models and low specs infrastructure (in Subsection 6.3 with Configuration 3).

We consider that each server acts as a hosting environment for a discipline/ role and runs CometCloud (in a more general context, a discipline can have multiple servers). The C4C framework is dynamically created at runtime, enabling sites to join or leave at any given time. Based on the use of CometCloud [4], each site has a master process that receives task requests (IFC objects to update or retrieve) from other sites, and is able to forward requests to other sites. Each site also has multiple worker processes that carry out actual task executions on locally available resources. We investigate how the aggregated time-to-complete changes when: (i) the number of IFC objects change, (ii) simultaneous client requests vary, (iii) the number of disciplines increase. From Figure3, the process starts when a client performs an update (changes one or more IFC object(s)) and finishes when the change can be observed at another discipline, requiring an object to be transferred from the client's local machine to the locally available master (on a data centre or server) and then to the node associated with the remote discipline.

Parameter	Default	Exp. 1	Exp. 2	Exp. 3
No. of disciplines	3	3	3	(2,3,4)
No. of objects	20	(1,5,10,20)	20	20
No. of concurrent requests	1	20	(1,2,3)	1

Table 2: Experimental testbed

To measure this process we use a time metric called "aggregated-time-to-complete" (ATTC) which is a sum of three individual times: (i)transfer time T_t, (ii) writing time T_w and (iii) overheads $T_{o/h}$ – as illustrated in figure 3, i.e. $ATTC = T_t + T_w + T_{o/h}$. The ATTC measurement is applied to the local client that initiates the requests

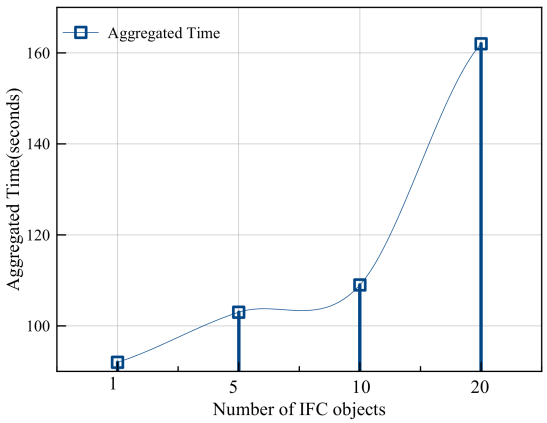
6.1. Low Spec. Infrastructure and Small Model

The experimental framework has the configuration provided in Table 2. The experimental setup for this scenario is presented in Configuration 1 identifying four virtual machines where each virtual machine(VM) has 1 CPU core at 3.2 GHz and an IFC model of size 3.44MB

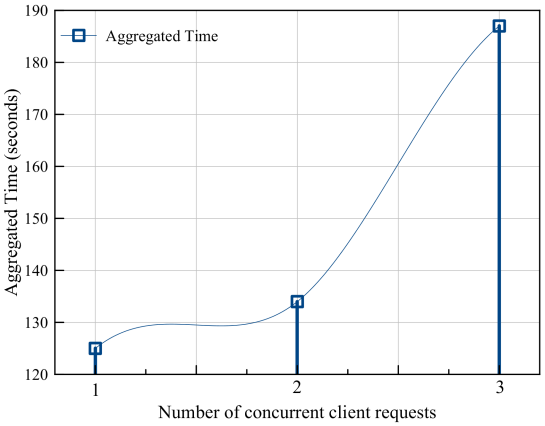
Experiment 1. : In this experiment we are interested to determine how the C4C system responds when dealing with an increased number of changes in the IFC model. The changes are identified by IFC objects which may be added, updated or deleted by the remote disciplines that are part of the overall C4C framework. Here we effectively measure the robustness of the system in dealing with events which may happen at different remote sites.

As illustrated in Figure 6a, the system reacts differently at different stages of the update process. The ATTC (Aggregated-Time-To-Complete) increases with the number of IFC objects that are being updated. The difference between updating a single object vs. 20 objects is determined by the overheads that appear at the master node of the local and the remote discipline, and the transfer time when migrating objects.

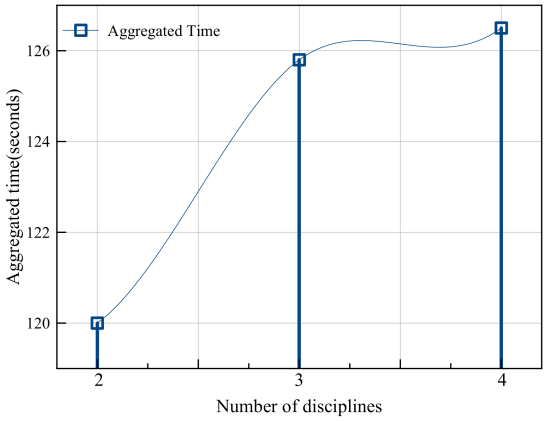
Experiment 2. In this experiment we investigate how a number of simultaneous client requests can impact ATTC, demonstrating how the C4C system can deal with an increase in the number of tasks received from remote clients. We increase the number of clients within the set [1,2,3]



(a) Aggregated-Time-To-Complete at different stages of objects updates.



(b) Aggregated-Time-To-Complete at simultaneous client requests.



(c) Aggregated-Time-To-Complete when varying the number of disciplines.

Figure 6: Preliminary experiments: Update model: a) Object updates, b) Concurrent time requests, c) Multiple disciplines

and the default number of requests (IFC objects to update) that a client can submit from a remote disciplines is 20.

Figure 6b shows an increase in ATTC with an increase in the number of concurrent requests submitted to the C4C system. In this case the discipline master not only has to look after internal tasks (received from the local client) but also deal with requests for IFC object updates submitted by remote clients.

Experiment 3. In this experiment we vary the number of disciplines and measure the impact on the system. When varying the number of disciplines that participate in a C4C project the number of IFC objects change as wells. In the AEC context this experiment demonstrates the scalability of the system, when there is an increase in the number of disciplines involved.

As illustrated in Figure 6c, the response time is relatively stable when the number of disciplines change. Increasing the number of disciplines requires a one-to-many communication, leading to a higher time-to-complete.

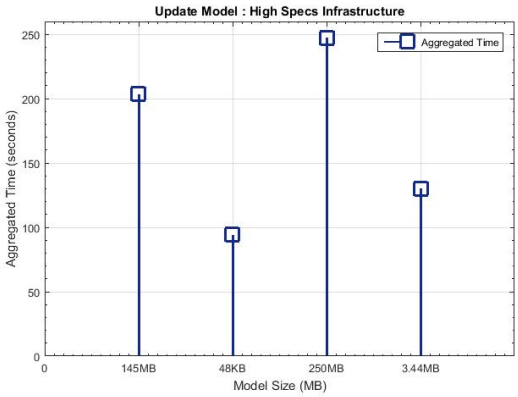
6.2. Performance analysis with high specs infrastructure and large models

In this second round of experiments we increase the size of the IFC models (illustrated in Figure 5b) and the configuration of the virtualized cluster-based infrastructure. In this scenario each server has 16 CPU cores at 3.2 GHz and 64GB of memory. The networking infrastructure is 1Gbps Ethernet with a measured latency of 14 ms on average. Each server runs Ubuntu 12.4 and Java 7. The experimental setup for this scenario is presented in Configuration 2.

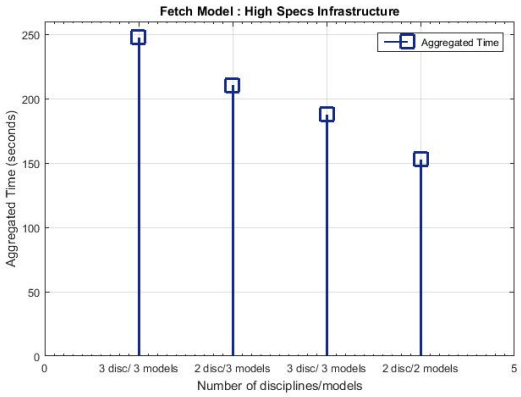
Within this new setup we investigate how the aggregated-time-to-complete changes when: (i) update model is called from a discipline client, (ii) fetch model is called from a discipline client, (iii) concurrent client requests take place within the system. The overall system workflow and time-to-complete can be found in Figure 3.

Experiment 4. In this experiment we investigate how the aggregated-time-to-complete varies with the size of the IFC models in the context of an update model request – in Figure 7a. The greater the number of objects in a model, the greater the increase in the time-to-transfer and time-to-write object data to disk, and associated overheads. Although relatively small models containing few objects lead to a relatively small time-to-write, in this case the aggregated-time-to-complete is influenced by the transfer time of the metadata across disciplines and the overheads at both the local and the remote disciplines.

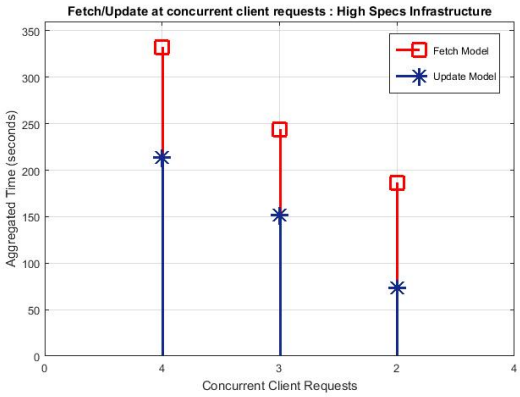
Experiment 5. Figure 7b illustrates time-to-complete when changing the number of disciplines and models in a fetch operation. In this experiment we consider that a disciplines can store multiple IFC models (i.e. three models distributed across two disciplines). When fetching from



(a) Aggregated-Time-To-Complete with different model sizes.



(b) Aggregated-Time-To-Complete with different number of disciplines and models



(c) Aggregated-Time-To-Complete with different concurrent client requests.

Figure 7: Update and Fetch with complex IFC models and high specs infrastructure: a) Update Model with model sizes, b) Fetch Model with multiple disciplines, c) Update and Fetch Model with concurrent client requests

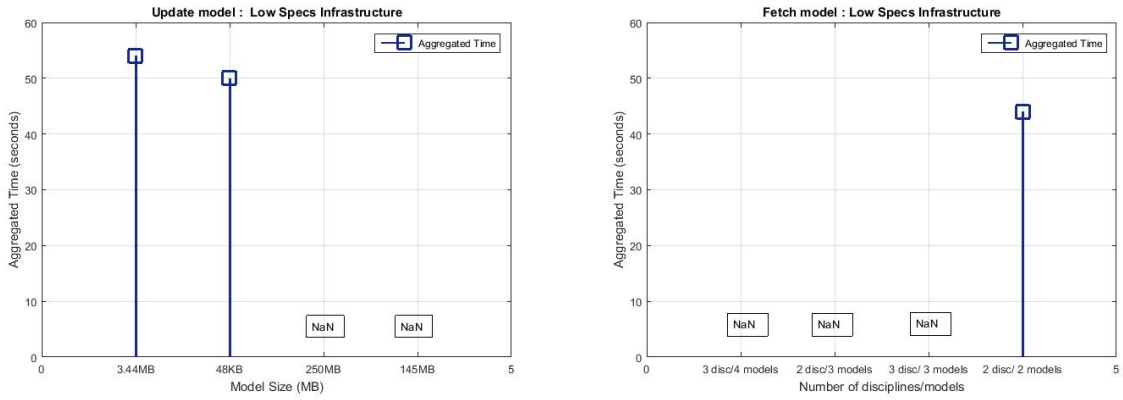
multiple disciplines there are several writing and parsing stages involved. It is observed that fetching four models from three disciplines determines a higher aggregated-time-to-complete than fetching three models from three disciplines. Fetching multiples models from a disciplines requires additional parsing and imposes overheads, hence the increase of aggregated-time-to-complete.

Experiment 6. In this experiment we investigate performance overheads associated with updating and fetching a model with a varying number of concurrent client requests. From Figure 7c we

can observe that fetching is always more demanding in terms of aggregated-time-to-complete. We can also note that aggregated-time-to-complete increases with the number of concurrent client requests. In the context of the update model request, the higher the number of client requests the greater the time-to-write, thereby leading to higher system overheads – due to multiple endpoints where model and metadata is saved. In the context of fetch model, fetching with concurrent requests involves simultaneous writes and data transfers across disciplines.

6.3. Low Spec. Infrastructure and Large Models

In this round of experiments we use the same size for the IFC models as in subsection 6.2 (illustrated in Figure 5) using our local cluster system and deploy four VMs, each with 1 CPU core at 3.2 GHz each. The experimental setup for this scenario is presented in Configuration 3. We change the experimental setup to a low specs infrastructure to test our system and observe how aggregated-time-to-complet changes when: (i) update model, (ii) fetch model, is called from a discipline client. The overall system workflow and time-to-complete can be found in Figure 3.



(a) Aggregated-Time-To-Complete with different model sizes.

(b) Aggregated-Time-To-Complete with different number of disciplines and models

Figure 8: Update and Fetch with complex IFC models and low specs infrastructure: a) Update Model with model sizes, b) Fetch Model with multiple disciplines

Experiment 7. In comparison to Experiment 4, we observe from Figure 8a that aggregated-time-to-complete changes when using a low specs computing infrastructure (refer to Configuration 3).

We use large IFC models and observe that update model can only be completed for smaller models of 3.44MB and 48KB. For models of size 145MB and 250MB respectively, we observe that update model cannot be completed. By using such large models over a low specs infrastructure of 1 VM with 1 CPU cores and 1GB of RAM, our system outperforms. This is mainly determined by the amount of RAM memory which is insufficient to deal with the number of IFC objects embedded in large models of 250MB and 145MB respectively. After additional testing we have determined that our system requires at least 4 GB of RAM to cope with large IFC models.

Experiment 8. In this experiment we investigate large IFC models executed over a low spec. computing infrastructure (refer to Configuration 3). From Figure 8b we observe that fetch model cannot be completed when distributing large models over the disciplines. The configurations considered (3 discipines/4 models , 3 disciplines/3 models, 2 disciplines/3 models) involve at least one large IFC model. Experiments 7 and 8 demonstrate performance of our system deployed over IBM Softlayer, with configurations similar to Configuration 2. Using virtual machines in the same cluster improves time-to-complete for both fetch and update model.

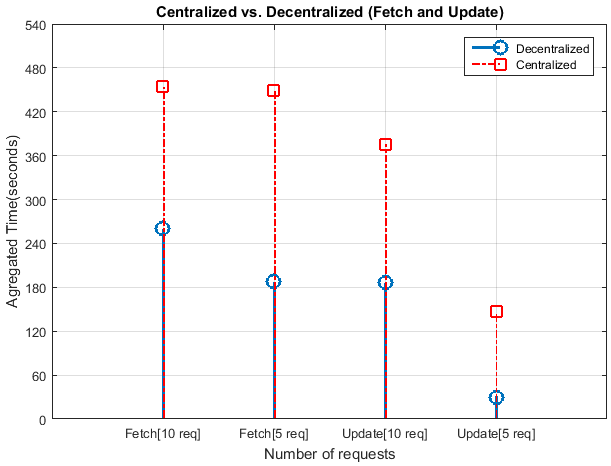


Figure 9: Fetch and Update – centralised vs. decentralised

6.4. High Spec. Infrastructure and Large Models

To demonstrate the benefits of our system to coordinate models which can be distributed over disciplines within a project we verify how the system behaves in comparison to the centralised approach. We use large IFC models (illustrated in Figure 5b) and the configuration of the virtualized cluster-based infrastructure. The experimental setup for this scenario is presented in Configuration 2.

Experiment 9. In this experiment, the number of concurrent requests are considered in: (a) decentralised and (b) centralised system. The decentralised context identifies a federated cloud distributing the IFC models. The centralised approach identifies a cloud system where the IFC models are stored and accessed locally. From Figure 9 we observe that the distributed federated cloud provides better time-to-complete than the centralised cloud. This is mainly determined by the multiple access points that the distributed cloud offers to users, where a model can be fetched and updated via multiple sites.

7. Conclusion

Companies involved in an AEC project often have their own data, hosted on local infrastructure, which they may be reluctant to move to a single data centre. Many existing Cloud solutions however require this, i.e. all data is moved to a single server or location, with subsequent access being controlled to various data sources at such a single location. The approach advocated in this work suggests that each company maintain its own data (on a local server, within a private Cloud environment, or on storage acquired from a public Cloud provider), without a need to migrate this data to a central site. Subsequently, an overlay-based Cloud environment is created, where all participants in a project can get access to a ”logically” shared data/compute space. This is achieved in this project by using the CometCloud system, which enables a number of different sites to be federated using the concept of a “CometSpace”. Access to data is through references that are maintained in CometSpace, with physical instances of data being maintained at their original point of creation. Data transfer is only undertaken if a user requesting the data has

suitable access rights to it. In this way, CometCloud differs from other Cloud systems (such as OpenStack), as it provides an important abstraction for supporting Cloud federation and "cloud bridging". We use the aggregated-time-to-complete metric to observe how the C4C system reacts in different scenarios. We observed that the time-to-complete is dependant on the number of concurrent client requests. We have also concluded that the number of disciplines that are part of an IFC project may also increase the time-to-complete. We demonstrate how this metric is influenced by the choice of the underlying infrastructure used to deploy the C4C system.

Acknowledgements

We are grateful for continued support from the Rutgers Discovery Informatics Institute (RDI2) on the use of the CometCloud system – in particular Javier Diaz-Montes and Manish Parashar. The C4C project is funded by EPSRC/InnovateUK and involves: Building Research Establishment, AEC3, Costain, Lee Wakemans Ltd, RIBA Enterprises, IBM and Cardiff University.

References

[1] OCCI – Open Cloud Computing Interfaces – available at: `http://occi-wg.org/`. Last accessed: February 2017.

[2] GENICloud - available at: `http://groups.geni.net/geni/wiki/GENICloud`. Last accessed: February 2017.

[3] Cloud Harmony - available at: `http://cloudharmony.com/`. Last accessed: February 2017.

[4] CometCloud Web site - available at: `http://nsfcac.rutgers.edu/CometCloud/`. Last accessed: February 2017.

[5] H. Kim and Parashar, M., CometCloud: An Autonomic Cloud Engine, in Cloud Computing: Principles and Paradigms, John Wiley & Sons, 2011, pp. 275-297.

[6] Ioan Petri, Tom Beach, Mengsong Zou, Javier Diaz-Montes, Omer Rana and Manish Parashar, "Exploring Models and Mechanisms for Exchanging Resources in a Federated Cloud", Int. Conf. on Cloud Engineering (IC2E), Boston, USA, March 2014. IEEE Computer Society Press.

[7] N. Carriero and D. Gelernter, "Linda in context", Communications of the ACM, vol. 32, no. 4, 1989.

[8] A.N. Toosi, R. N. Calheiros, and R. Buyya. Interconnected cloud computing environments: Challenges, taxonomy, and survey. ACM Comput. Surv., 47(1):7:1–47, May 2014, http://dx.doi.org/10.1145/2593512.

[9] Adel Nadjaran Toosi, Rodrigo N. Calheiros, Ruppa K. Thulasiram and Rajkumar Buyya, "Resource Provisioning Policies to Increase IaaS Provider's Profit in a Federated Cloud Environment". In Proceedings of the 2011 IEEE Int. Conf. on High Performance Computing and Communications (HPCC '11). IEEE Computer Society, Washington, DC, USA, pp. 279-287, 2011.

[10] Tuple Space-based abstraction - details available at: http://en.wikipedia.org/wiki/Tuple_space. Last accessed: April 2014.

[11] British Standard 1192:2007 – "Collaborative production of architectural, engineering and construction information, Code of practice". ISBN: 978 0 580 58556 2, British Standards Institute (BSI), January 31, 2008.

[12] David Bryde, Mart Broquetas, Jargen Marc Volm, The project benefits of Building Information Modelling (BIM), International Journal of Project Management, Volume 31, Issue 7, October 2013, Pages 971-980, ISSN 0263-7863,

[13] M. Latham, Constructing the Team: Final Report of the Government/Industry Review of Procurement and Contractual Arrangements in the UK Construction Industry., London: HMSO., 1994.

[14] J. Egan, Rethinking construction, Department of the Environment, Transport and Regions.

[15] A. Dainty, D. Moore, M. Murray, "Communication in Construction: Theory and Practice", Abingdon, Oxon: Taylor & Francis, 2006.

[16] Becerik-Gerber, B., Jazizadeh, F., Li, N., and Calis, G. (2012). "Application Areas and Data Requirements for BIM-Enabled Facilities Management. J. Constr. Eng. Manage., 138(3), 431–442.

Author Bios

Ioan Petri is a Research Associate in the School of Engineering at Cardiff University.

Omer Rana is Professor of Performance Engineering in the School of Computer Science & Informatics at Cardiff University. He leads the Complex Systems research group.

Tom Beach is a Lecturer/Assistant Professor in the School of Engineering at Cardiff University.

Yacine Rezgui is a Building Research Establishment (BRE) Chair in 'Building Systems and Informatics'. He is the Director of the BRE Trust Centre of Excellence in Sustainable Engineering.