2015 3rd International Conference on Future Internet of Things and Cloud

Distributed cloud bursting model based on peer-to-peer overlay

Andrii Zhygmanovskyi and Norihiko Yoshida
Department of Computer Science
Saitama University
Saitama 338-8570, Japan
Email: {andrew,yoshida}@ss.ics.saitama-u.ac.jp

***Abstract*—In this paper, we present an approach to building cloud bursting architecture based on the peer-to-peer infrastructure for managing services. Proposed approach is designed to address various issues of interconnecting several clouds, problems of resource provisioning, service deployment and provisioning in the hybrid cloud. To ensure robustness of our system we use peer-to-peer overlay, which was proposed by us in previous papers. Scalability of the approach is attained due to flexibility of service discovery mechanism, decentralized architecture and modular approach, which allows to leverage existing components. We argue that our approach present viable solution for managing abrupt peaks in the load and keeping service provider's QoS and SLA requirements on the desired level.**

***Keywords*-cloud computing; cloud bursting; distributed systems; peer-to-peer; service provisioning; intercloud; multicloud**

I. Introduction

Cloud computing emerged as a novel approach allowing anyone to quickly provision a large scale IT infrastructure that can be completely customized to user needs on a pay-per-use basis. Recently, there is an increasing research effort concerning concomitant use of two or more cloud services to minimize the risk of widespread data loss or downtime due to a failure in a cloud computing environment. Although the terminology might not be fixed yet due to the novelty of the research domain, most researchers tend to use the term *intercloud computing* which is formally defined as "a cloud model that, for the purpose of guaranteeing service quality, such as the performance and availability of each service, allows on-demand reassignment of resources and transfer of workload through the network of cloud systems of different cloud providers based on coordination of each consumers requirements for service quality with each providers SLA and use of standard interfaces“ [1]. This model, in turn, can be subdivided into two more specific categories — *cloud federation* and *multicloud* — according to the kind of interaction between cloud providers. Composition of two or more different cloud infrastructures (for example, a private and a public cloud) falls into separate category called *hybrid cloud*. The most common scenario, when such infrastructure emerges, is the the usage of external cloud resources when local ones are insufficient, which is called *cloud bursting* — an approach that is drawing more attention each year. However, due to being an inherently distributed system, cloud bursting potential becomes limited when such vital tasks as resource management and allocation, service provisioning and life cycle management are implemented in a centralized way, usually utilizing some kind of brokering architecture, which introduces multiple issues, such as emerging of single point of failure, performance bottlenecks, network congestion and synchronizing problems. This leads to the need of making at least part of the cloud bursting solution decentralized, therefore increasing its robustness, scalability and fault tolerance. Considering our previous efforts in applying peer-to-peer based approach to the problem of service sharing and discovery [2], we decided to investigate its possible applications within the intercloud research domain.

Principal contributions of this paper include a) a proposal for cloud bursting architecture that facilitates decentralized management of cloud resources and provides end-users with fault-tolerant and reliable services in large, autonomous, and highly dynamic environments; b) application of previously proposed service sharing and discovery approach based on peer-to-peer overlay to streamline the execution of single jobs and entire workflows in the cloud; and c) introducing a modular framework for composing cloud bursting solutions, allowing adoption of various standards and tools.

The rest of the paper is organized as follows. Section II describes research background and outlines related work. Section III presents proposed architecture for decentralized cloud bursting and describes the process of service provisioning. Considerations and metrics for framework evaluation as well as our vision regarding comparative analysis with other intercloud and cloud bursting systems are presented in Section IV. Conclusions and further research directions are given in

Section V.

II. Research background and related work

Wide adoption and growing reliance on cloud technology are among the main reasons for situations where one cloud (or cloud provider) becomes not enough. Such cases include a) outages within cloud provider premises; b) insufficient geographical distribution of cloud provider resources; c) added complexity of migrating the infrastructure from one cloud provider to another, and as a result, d) the possibility of vendor lock-in. Yet, the biggest problem that emerge from using single cloud provider is the limits in scalability, which most often manifests itself in form of poor handling of abrupt fluctuations in the load, when permanent scaling out is not economically reasonable, and at the same time peaks in the load must be handled as prompt as possible. Flash crowds can be named as a prominent example. Thus, proposed architecture is designed to deal with the following problems: a) general approach to interconnecting several clouds; b) resource provisioning in the hybrid cloud; c) service deployment and provisioning in the hybrid cloud. Virtually all the solutions that we are aware of introduce certain centralized component that play the role of a broker, which usually combine multiple responsibilities, such as a) acting a marketplace where clouds can sell or advertise resources [1]; b) mapping user requests to cloud resources [3]; c) maintains the registry of collaborating clouds' services [4], etc. We argue that existence of such component makes the system vulnerable to all kind of intrinsic failures to which centralized systems are susceptible to, as already described in the introduction.

Current intercloud research efforts can be considered still on its early stage and include Contrail [3], mOSAIC [5], OPTIMIS [6] and RESERVOIR [7]. More detailed analysis of intercloud frameworks can be found in [1]. As for cloud bursting research, the results here are sparse and include an architecture described in [8], investigation about decision support model for cloud bursting in [9], and reference design for cloud bursting with special attention to the problems of trust and security in [10].

III. Proposed model

At the beginning of the description of our model, we provide details about participating entities. In case of cloud bursting scenario, they usually include *service provider* (SP), which provides one or several services, which it operates on premises using private cloud, and *cloud provider* (CP), which is a company that offers a cloud computing solution in the form of IaaS, PaaS or SaaS (or any combination thereof) to other businesses or individuals. Cloud bursting scenario typically involve several CPs, either on the evaluation stage, during which SP choose the CP it would use for cloud bursting, or during actual multicloud stage if SP uses several public clouds. Chosen CP is called *cloud bursting target*. In our case, scenario also includes another role, called *client*, which represents an entity that has some work to perform in the cloud (either in the form of the workflow instance or a single job request). While client in general might be a separate entity, in most scenarios SP acts in this role. Lastly, we will call a request from the client a *job request* from now on, but it should be noted that this definition differs from the similar terminology that exists in Grid systems.

A. Service descriptions and service queries

Services provided by SP are described using (a, v)-graph notation, introduced in [2]. We call the resulting (a, v)-graph a *service* (a, v)*-graph*, or simply *S-*(a, v)*-graph*. Next, each virtual machine instance in the cloud is characterized by its own (a, v)-graph based description, which includes hardware characteristics such as CPU information, maximum available memory, maximum available hard disk space, operating system type and version, available licenses and installed certificates, etc. This graph is built similarly to the S-(a, v)-graph and is called *instance* (a, v)*-graph*, or simply *I-*(a, v)*-graph*. Virtual machine instances leased from the external CP add more (a, v)-pairs to the descriptions of the services they host. Those (a, v)-pairs contain information such as maximum lease time, lease slot duration or resource price, and form so called *leased resource* (a, v)*-graph* or simply *L-*(a, v)*-graph*. Finally, service, instance and leased resource (a, v)-graphs are merged into one graph called *full-*(a, v)*-graph*, which is stored at the overlay node. Similarly to the original paper, full-(a, v)-graph contains the node that holds low-level routing information about the instance.

Job request is a crucial part of cloud bursting approach, since it acts as the mechanism which determines the status of the internal cloud and verifies to what degree provisioned services fulfill the established service level agreement and other auxiliary requirements. In case the job that is being submitted to the cloud cannot be executed with only internal resources, leasing additional resources from external cloud must be performed. In our model we construct such mechanism using the set of *service queries* $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_s\}$ that formally describe each job request J and return the set of available service instances $\mathcal{I}$ that are provided by either SP or one of the CPs that currently act as a cloud bursting target. The service query is expressed in a format introduced in [2].

The node, which handles the query, is called *brokering node*. Detailed description of query processing also may be found in [2]. Apart from service properties, query can include special terms, called *provisioning policy terms*, which help regulate the rate and scale of overlay growth. Examples of these terms are cost restricting terms and queue wait time.

B. Overlay formation and scaling

Next, we describe the procedure of forming a peer-to-peer overlay which is the pivotal element in the proposed architecture. It is formed with cloud virtual machine instances as nodes, each of which has at least one service deployed, however some services may be deployed on several instances as well, depending on the capabilities they require and estimated initial demand. We consider it reasonable to assume that services of SP form an overlay network of considerably large amount of nodes, since if the opposite were true SP most probably would not need to resort to building cloud bursting solution. The form and the scale of the overlay is defined by the combination of the following scenarios:

1) *Initial configuration scenario:* Starting point of building cloud bursting architecture. Here SP first deploys its services into own private cloud with minimum needed amount of instances for each service, therefore establishing sufficient infrastructure for executing necessary jobs at the moment.
2) *Vertical scaling scenario:* This scenario consists of expanding or contracting (increasing or decreasing the number of instances) within the bounds of the SP private cloud and does not incur extra-corporate expenses.
3) *Horizontal scaling scenario:* This scenario represent the actual cloud bursting process: temporary leasing virtual machine instances from cloud bursting target and eventual contracting to the original state of having only private cloud instances. Choosing one or several CPs that will act as a cloud bursting target is also performed here, but can be seen as optional, because in the real-word scenario SP will not perform full CP selection process each time the horizontal scaling occurs.

By utilizing peer-to-peer overlay, we achieve certain level of homogeneity, which allows us to abstract from actual nature of scaling scenario (whether it is vertical or horizontal), since in both cases new virtual machine instances are represented as overlay nodes with corresponding service descriptions that are linked to the actual instance routing information. Given this model, each job request will result in one of the following situations:

1) Requested service is available and job request satisfies restrictions introduced by resource provisioning policies.
2) SP infrastructure lacks either power (expressed in CPU, memory etc) or capabilities (expressed in operating system kind, software, license, protocols support, certification etc). In this case brokering node can put the request in the job request queue (performing periodical re-querying) or scaling can be performed, resulting in increased amount of instances and/or adding required capabilities.

C. Platform components

In this subsection we describe the components which compose proposed solution. Due to modular approach to designing and implementing the solution, components are loosely coupled, allowing replacing or refactoring them if needed. This approach also makes possible using a wide range of third-party solutions for queues or databases. We argue that proposed solution decentralizes most components, that usually appear centralized in other publications related to cloud bursting or multi-cloud organization, as shown in Table I.

Table I
COMPONENTS DECENTRALIZED IN PROPOSED MODEL.

Component	Examples
Resource pool	FCM Repository [11], Cloud Computing Resource Catalogue [12]
Service request processor	Federation Runtime Manager [3], Generic Meta-Broker Service [11], Resource Broker [5]
Resource provisioner	Cloud Broker [11], Cloud Optimizer [6], Client Interface [5]
Multicloud broker	Primary Cloud Provider [4], Cloud Coordinator [13], Intercloud Exchange [12]

The components that are used in the proposed model are as follows. Dependencies between components are shown in the Figure 1.

Job request processor: Gateway component which accepts a job request from a client. The job request is put into the job request queue, and then a service query is formed and sent to the service query processor, which returns the result together with further action. Depending on this action, job request processor either

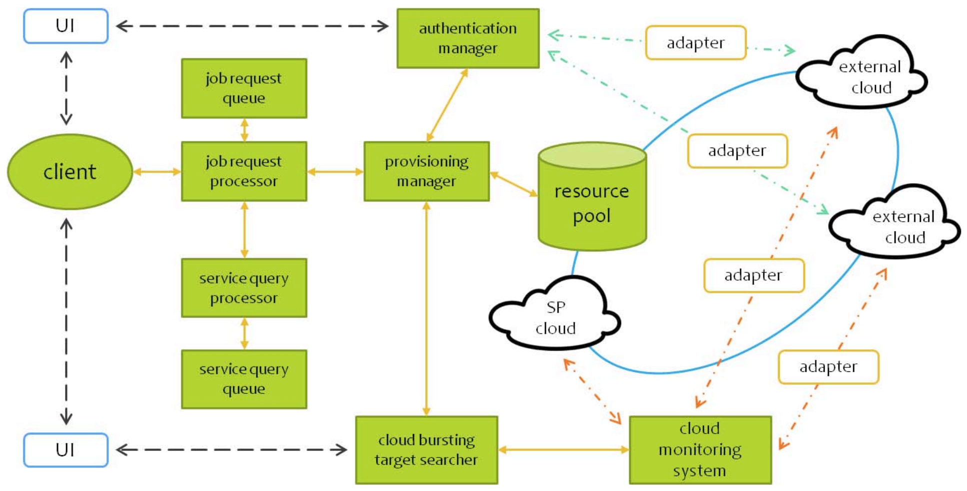

Figure 1. Model components.

contacts provisioning manager and eventually return available instance(s) information to the client or repeats service query later. When job is finished, its results (if they are present) are returned to the client. In many cases, clients may receive not the actual result value, but a set of pointers to the component where the result data is stored.

Service query processor: Component which is responsible for processing the service query, which was formed by job request processor based on the job request from a client. Since service query processing itself is a multi-stage process, the component performs the following functions:

- Accepts service query and puts it to the service query queue.
- Processes the query according to procedure described in [2].
- After getting the query result, returns it to the job request processor along with the action that is supposed to be taken (such possible actions are described in detail in Subsection III-B).

Since proposed solution is distributed, any overlay node can act as job request processor and service query processor for the incoming query. In practice, actual node is chosen using round-robin algorithm in order to distribute the load and eliminate unnecessary large request queues. Furthermore, while the architecture actually does not prescribe that those two components must be located at the same overlay node, it seems reasonable to do so for given job request and service queries associated with it in order to decrease network traffic and improve reliability.

Job request queue: Component which holds job requests with their respective service queries being processed or scheduled to do so. Physically, the queue is organized as a distributed queue where master node corresponds to the overlay node that acted as job request processor, and several other nodes in the overlay act like a slave nodes with duplicate queue instances.

Service query queue: Component which holds service queries which are currently being processed. This queue can be organized in a distributed way similar to job request queue, but in reality due to reliability concerns it is coupled with service request processor and located at the same node.

Provisioning manager: Component which is responsible for provisioning of resources in case the current ones are not sufficient to process a job request. It is contacted by job request processor in case service request result indicated that new instances must be created for given job request. On this stage, provisioning manager becomes the point from where cloud bursting process starts in case local resources are not enough. First, it contacts cloud bursting target searcher component to obtain the information about most suitable cloud bursting target. Having established the target, provisioning manager sends the request to external authentication manager, which provide necessary credentials and other authentication data needed to start provisioning process from external cloud. It uses appropriate adapter components to interact with external CP, obtain routing information for leased resources and return it to job request processor.

Resource pool: Since most cloud providers offer billing plans based on fixed time blocks (e.g. X dollars for Y hours of uptime), it is reasonable to maintain a pool of resources that once was leased from CP with job that requested them is finished but the dedicated time slot is not over yet, so they can be reused. In contrast to most centralized solutions, this pool does not exist as a standalone component — the nodes just remain as the part of the overlay with respective services deployed. Nodes are removed from the pool (and therefore from the overlay) based on their L-(a, v)-graph information. Releasing these resources can be managed in the distributed way when overlay nodes check their neighbors and send corresponding ”release” message to provisioning manager.

External authentication manager: Component which stores and manages various authentication information (credentials, authentication tokens, security certificates, etc.), that are necessary to successfully perform resource leasing from external CP. This component is contacted by provisioning manager during cloud bursting phase. This component uses adapter components for interacting with external CP.

Cloud bursting target searcher: Component which provides information about cloud bursting target when horizontal scaling is performed. Possible cloud bursting targets are first listed in the initial configuration of the component in the order of preference. This configuration is created based on open and available data about CPs, which include pricing plans, available geographical locations, supported platforms and services, etc. In addition, data from cloud monitoring system component is used to change this initial preferences order. The design of proposed solution also allows the administrator to participate in cloud bursting target selection process by manually performing selection via GUI.

Cloud monitoring system: Component which collects, aggregates and stores diagnostic and monitoring data obtained from hybrid cloud instances. While every CP provides its own cloud performance monitoring system (for example, Amazon CloudWatch [14] or Cloud Monitoring [15]), they are not designed to be cloud-agnostic, which renders their usage in multicloud scenarios difficult. Notwithstanding that, there are some systems which are oriented for multicloud usage [16] and some of them can be used in our solution. While it does not seem reasonable to make such system truly decentralized, it can benefit from certain techniques such as replication. Monitoring system output can be used to decide whether we need to perform scaling in some particular situation.

Adapter components for interacting with external CP: While there are some efforts to provide a common standard API for the cloud [17], at the moment each cloud provider still uses vendor-specific interfaces for cloud programmatic access and communication with their cloud resources. This brings forward the need to maintain separate driver components for each possible cloud bursting target. In our solution, those components constitute the part of the application, that is deployed on every instance.

IV. Considerations for evaluation and comparative analysis

In this section we will present metrics and other considerations for evaluation and comparative analysis of our proposed approach which is done as a part of future research on this topic. In order to test behavior and performance of the system we are currently building a simulator for cloud bursting scenario, which contains all components described in Section III-C and mocks necessary interfaces that are present in real cloud setting.

First, we propose several basic performance metrics as follows.

- *Average time of job request in job request queue*. This metric shows the overall responsiveness and effectiveness of the system.
- *Average amount of service query executions (reformulations) for given job request*. This metric reflects the responsiveness of the system and its sensitivity to the private cloud overload.
- *Average lifetime of a resource in the resource pool*. This metric estimates the size of the hybrid cloud. It can be measured separately for SP and external CP instances to show how heavily SP depends on leased resources.
- *Average cost per job request*. This is a metric that shows how costly is job request processing from SP perspective.
- *Network load*. This metric shows how efficiently the system is using network resources.

Next, there is a need for comparing proposed solution with other research results in this area. While there already are multitude of comparative analysis research for traditional cloud systems, intercloud and cloud bursting systems presents certain complexity in this aspect because of novelty of the topic. The following shows how our solution fall into intercloud systems taxonomy introduced in [1]: a) peer-to-peer multicloud library *(architecture)*; b) SLA based *(brokering approach)*; c) singular/periodical jobs, compute- and data-intensive interactive application *(application type)*; d) geolocation, pricing, legislation/policy, local resources *(awareness)*.

As a part of future research, we plan to perform comparison based on the following characteristics: a) level of adhering to client SLA requirements; b) how optimal is the use of SP costs; c) level of cloud monitoring and how far this data is leveraged to optimize bursting target selection; d) how system reacts to abrupt/planned peaks in the load; e) robustness in case of partial cloud failure. In cases when these and other criteria are hard to measure using quantitative methods, we intend to perform detailed analysis to show pros and cons of our approach from the qualitative aspect.

V. Conclusions and future research

In this paper, we presented a distributed approach of building cloud bursting system based on peer-to-peer overlay originally designed for service sharing and discovery. The main advantages of proposed approach are a) introducing decentralized brokers, which eliminates single point of failure and increases robustness of the system; b) eliminating separate resource pool component and instead forming it in form of cross-cloud peer-to-peer overlay, therefore making the system highly scalable; c) utilizing well-known DHT, which guarantees the correctness and soundness of underlying peer-to-peer overlay; and d) modular framework architecture, which allows using wide range of existing queue or storage components.

As for prospective directions for future research, we intend to automate decision making process for choosing most suitable cloud bursting target, possibly by using detailed analysis of historical data to build a knowledge database, which then could be used to infer optimal choice. Another direction is adapting proposed model for hosting business workflows, since we believe that our approach is suitable for many scenarios that emerge in workflow orchestration and execution due to its distributed nature.

References

[1] N. Grozev and R. Buyya, "Inter-Cloud Architectures and Application Brokering: Taxonomy and Survey," *Software: Practice and Experience*, vol. 44, no. 3, pp. 369–390, 2012.

[2] A. Zhygmanovskyi and N. Yoshida, "Cloud service provisioning based on peer-to-peer network for flexible service sharing and discovery," *Journal of Computer and Communications*, vol. 2, no. 10, pp. 17–31, 2014.

[3] E. Carlini, M. Coppola, P. Dazzi, L. Ricci, and G. Righetti, "Cloud federations in Contrail," in *Euro-Par 2011: Parallel Processing Workshops*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2012, vol. 7155, pp. 159–168.

[4] M. M. Hassan, B. Song, and E. nam Huh, "A market-oriented dynamic collaborative cloud services platform," *Annales des Télécommunications*, vol. 65, no. 11-12, pp. 669–688, 2010.

[5] D. Petcu, B. D. Martino, S. Venticinque, M. Rak, T. Máhr, G. E. Lopez *et al.*, "Experiences in building a mOSAIC of clouds," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, no. 12, 2013.

[6] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri *et al.*, "OPTIMIS: a holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, January 2012.

[7] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente *et al.*, "The Reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 535–545, July 2009.

[8] M. Mattess, C. Vecchiola, S. K. Garg, and R. Buyya, "Cloud bursting: Managing peak loads by leasing public cloud services," in *Cloud Computing: Methodology, Systems, and Applications*. CRC Press, Taylor and Francis Group, LLC.

[9] M. Lilienthal, "A decision support model for cloud bursting," *Business & Information Systems Engineering*, vol. 5, no. 2, pp. 71–81, 2013.

[10] R. Yeluri and E. Castro-Leon, "A reference design for secure cloud bursting," in *Building the Infrastructure for Cloud Security: A Solutions View*. Apress.

[11] A. Marosi, G. Kecskemeti, A. Kertesz, and P. Kacsuk, "FCM: an architecture for integrating IaaS cloud systems," in *Proceedings of the Second International Conference on Cloud Computing, GRIDs, and Virtualization*, Rome, Italy, 2011, pp. 7–12.

[12] D. Bernstein, D. Vij, and S. Diamond, "An intercloud cloud computing economy — technology, governance, and market blueprints," in *Proceedings of the 2011 Annual SRII Global Conference*, ser. SRII '11. IEEE Computer Society, 2011, pp. 293–299.

[13] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing, Volume Part I*, ser. ICA3PP'10. Springer-Verlag, 2010, pp. 13–31.

[14] Amazon CloudWatch. [Online]. Available: http://aws.amazon.com/cloudwatch/

[15] Google Cloud Monitoring. [Online]. Available: https://cloud.google.com/monitoring/

[16] G. Aceto, A. Botta, W. De Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, June 2013.

[17] J. K. Wang, J. Ding, and T. Niu, "Interoperability and standardization of intercloud cloud computing," *The Computing Research Repository*, vol. abs/1212.5956, 2012.