A Coordination-based Brokerage Architecture for Multi-Cloud Resource Markets

Sarah Aldawood, Frank Fowley
IC4 & School of Computing
Dublin City University
Dublin 9, Ireland

Claus Pahl, Davide Taibi
Faculty of Computer Science
Free University of Bozen-Bolzano
Bolzano, Italy

Xiaodong Liu
Inst for Informatics & Digital Innovation
Edinburgh Napier University
Edinburgh, U.K.

***Abstract*—With an increasing number of service providers in the cloud market, the competition between these is also increasing. Each provider attempts to attract customers by providing a high quality service with lowest possible cost and at the same time trying to make profit. Often, cloud resources are advertised and brokered in a spot market style, i.e., traded for immediate delivery. This paper proposes an architecture for a brokerage model specifically for multi-cloud resource spot markets that integrates the resource brokerage function across several cloud providers. We use a tuple space architecture to facilitate coordination. This architecture supports specifically multiple cloud providers selling unused resources in the spot market. To support the matching process by finding the best match between customer requirements and providers, offers are matched with regard the lowest possible cost available for the customer in the market at the time of the request. The key role of this architecture is to provide the coordination techniques built on a tuple space, adapted to the cloud spot market.**

***Index Terms*—Cloud Resources Market, Tuple Space, Resource Brokerage, Spot Market, Cloud Brokerage Architecture**

I. INTRODUCTION

In the cloud, computing resources such as hardware, network, storage, software, business service are available when needed and charges are based the rate of usage [1,3]. Many companies provide the same cloud services to customers, which increases the competition between the companies in the cloud. A trend in this market place in recent years is trading computing resources in a stock market style. Dynamic spot markets emerge trading resources for immediate delivery. These can be supported 'as-a-service' that is delivered immediately or within a short period of time. Our contribution in this context is twofold:

- Firstly, we analyse cloud resource market aspects and determine principles and features of a multi-provider cloud resource spot market.
- Secondly, we present a coordinated broker architecture for a resource spot market that allows different providers and consumers to participate in a brokering process.

Cloud spot markets require coordinated brokerage [2]. This has to consider multiple providers with a range of resources offered to a number of potential consumers. We explore a model that prioritises consumer benefits, to address a limitation of current single-provider markets that favour the provider. The focus of the paper, however, is on architectural requirements for such a market mechanism.

This requires different providers of resources and different potential consumers to be integrated in an easy way. We also need a mechanism to map the resource brokering with offers and requests easily. We propose a tuple space architecture as the coordination backbone of a spot market brokerage solution, which aims to solve the communication overhead problem to achieve scalability and can be used and tailored to support fine-grained coordination activities. A tuple space is a coordination model for parallel processing and data sharing proposed in the Linda model. Its architecture is suitable for communication for cloud service computing [1,5]. Cloud-specific is a need for coordination by 3rd-party service, with specific type of auctions (spot market). This results in an efficiency/scalability requirement for short responses. In our case, qualified coordination meets the 3rd-party brokering automation.

Our tuple space architecture extents common tuple space implementations, such as the Javaspace package, in order to deal with the brokerage features required. Particularly the matching process in fine-grained spot market models needs to be addressed, where resources differ in their technical properties, but also availability, location and pricing properties. We developed an additional feature for the tuple space. We evaluate the effectiveness of the proposed architecture through an experimental analysis. Tuple spaces are provide the required scalability [1,5].

Section 2 analyses how cloud services are brokered and discusses principles of cloud resource spot markets. The architectural framework is described with its matching support in Sect. 3. Sect. 4 covers the evaluation. Related work is covered in Sect. 5, before ending with conclusions.

II. BACKGROUND &ANALYSIS

For cloud resource markets, like spot markets, we analyse market solutions and determine broker architecture principles.

A. Market-related Properties of Cloud Computing Resources

Cloud computing can be defined as a large-scale distributed computing driven by economies of scale, in which a pool of abstracted, virtualized, dynamically scalable and managed resources such as computing power, storage, platforms, and services are delivered on demand to external customers [3]. A public cloud comprises of hardware and software services

made available a pay-as-you-go manner as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [7].

Quality of service (QoS) is an important factor [7]. Therefore, the decision associated with choosing service providers are based on the *price*, the *QoS* guarantees it provides to customers, and the satisfaction of the promised guarantees. QoS means the ability to provide different priorities to different applications, users, or data flows or to guarantee a defined level of performance to a data flow. Setting the costs of the services at a high price could potentially led to an increase in the cloud providers profits while at the same time might decrease the number of consumers, whereas setting low prices might attract more consumers and reduce provider profits. Similarly, satisfying high QoS levels would increase demand for services. Therefore, price and QoS level set by providers are playing an important apart in a cloud resources market. Architecture for cloud market with a negotiation mechanism (more general than auction mechanism) have been presented [28,29].

The pricing policies determine resource allocation types, resource rates and other costs. Also, each provider has different ways of allocating resources to a machine. Architecture for cloud market with a negotiation mechanism (more general than auction mechanism) have been presented [28,29].For example, Amazon and GoGrid have many machine types, so a customer can choose a resource set, not necessarily allowed to customize a machine. Other providers, e.g., CloudSigma, allow consumers to customize the resource of a machine by selecting the number of cores and the size of RAM as they need [8].

B. Principles of Cloud Spot Markets

A **market** is an environment that supports buyers and sellers with rules of interaction between them, which are set through the market mechanism. The **market place** should offer a set of different services to attract customer to the market and making the market service convenient, secure and low-risk. For example, pay-as-you-go is the most popular pricing schemes offered by public cloud providers. This scheme involves customers purchasing units of computing time and being charged hourly at a fixed price set, which might result in unused resources. Thus, a provider can sell unused resources at a reduced price rather than letting them go to waste.

This can be achieved through a **spot market** with an *auction mechanism* designed by the cloud provider [10]. The spot market is a *securities market*, on which goods are sold for cash and delivered immediately or within a short period of time with immediately effective contracts.

For computing resources, a **spot market** *trades computing resources* with a 'bid or ask' mechanism. All parties publicly announce the maximum price they are willing to pay for the product or service, which is usually set by the customer, and the minimum price they are willing to sell for is set by the providers of the resource. The **spot bids** are sorted in decreasing order of price in the spot bid queue, while the **spot asks** are sorted in increasing order of price in the spot ask queue [10]. The idea of the cloud spot market is that providers sell their unused resources during some specific low-demand times (weekends or at night time). Generally, this allows customers to bid in an auction system that is designed/managed by cloud providers. The customer who bids the higher price wins the bid. Usually, the auction system updates the price regularly, for example every five or ten minutes. Some providers remove the resource from the customer when the price becomes higher than the customer's bid, while others do not, unless the time they guarantee the resource allocated to the customer has ended.

The spot market is suitable for applications that have no real-time availability constraints, such as large data analysis, scientific computing and financial modelling, because usually the resources allocated in this market are not reliable [11]. *Auction systems* facilitate providers by setting a price for some resource types at a specific time and the provider who offer the lowest price during the time of customer's demand win the competition. A sample auction mechanism is the Amazon EC2 spot market, which lets customers bid for IaaS instance hours [12]. However, Amazon does not reveal information about their auction mechanism and the calculation of the spot price.

There are two points to be considered relating to resource pricing that providers should concentrate on: (i) the actual cost of resource and (ii) the profit maximization. There are many factors that influence pricing in the cloud. The most common influencing factor is the *initial cost*, i.e., the amount of money that a service provider spends to buy resources. The second consideration is the *lease period*, which is the period for which customers will lease resources from providers. Also, the better the QoS that is offered, the higher the price will be. Further influencing factors are the *age of resources* and the *cost of maintenance* that the service provider spends over a long time on maintaining and securing the cloud. There are other factors affecting the total cost: *computing resources allocated to the instance* (such as RAM, CPU, Disk, etc.), *geographical location* as different locations will be subject to different prices even by the same provider, *operating systems*, *minimum commitments*, reliability of service and *data traffic* [8].

C. Sample Spot Market Mechanisms

We now review some concrete offerings to illustrate the market factors for cloud resources. More comprehensive surveys, in particular concerning pricing models, have been carried out, e.g. in [13]. We select two widely used to extract architectural requirements, which is the key concern here.

1) Amazon Web Service EC2

Amazon Elastic Compute Cloud (Amazon EC2) is a resizable compute capacity. Amazon EC2 has 18 types of pre-defined 'customized' machines. These instances differ based on the number of cores, size of memory and other performance parameters. These types are priced differently depending on the location of the datacentre and the operating systems on the server. There are three different options for EC2 instance purchasing: on-demand, reserved instance and spot instance, which differ on the reservation period, availability and reliability [10,13]. There are further costs for data transfer and IP addresses. The different options for instance purchasing are:

1. **On-Demand instance**: Customers pay for compute capacity by the hour with no long-term commitments. The total price depends on how long the customer runs the machines. The benefit here is that the customer is relieved

from the complexities of planning, purchasing, and maintaining hardware. The resources are allocated once the datacentre contains enough own resources to run it.

2. **Reserved Instances**: This type of pricing is 75 percent less than on-demand instance pricing, because it is subject to a long term commitment. The customer has to pay the reserved instance fee upfront for the contract period, plus an hourly fixed rate. The minimum charge is the upfront fee, even if the customer never runs the machine. In addition, the marketplace for reserved instances is available to the customer to sell reserved instances when their need changes, or sell capacity for projects that end before their reserved instance term expires.
3. **Spot Instances**: Customers bid on unused capacity. Resources are allocated to customers when their bid exceeds the current spot price. The spot instances price changes periodically (e.g., every 5 min) based on supply and demand. The guarantee of the resource allocated to the customer depends on the bidding price. Therefore, when the spot price rises beyond the bidding price, resources would not be available and machines terminate (customers are not charged for that period). The drawbacks of spot instances are unreliability and lack of suitability for applications with real-time availability needs.

2) CloudSigma

CloudSigma is an IaaS provider. It differs in the pricing flexibility and resource allocation policy. Customers can customize their machine – minimum and maximum resource configuration like number of cores, size of memory and storage. It has two pricing schemes depending on commitment: subscription pricing and burst pricing. The total price for both schemes is calculated by the amount of resources selected by the customer [8]:

- Subscription pricing: customers choose resources and pay for a certain period before the service starts, with a minimum contract length.
- Burst pricing: a pay-as-you-run plan like Amazon on-demand pricing. The price is updated every five minutes, but without machine termination, even if the price is raised. Thus, these features make this scheme reliable.

D. Spot Market Architecture Requirements

Many public cloud providers provide services at fixed prices for longer periods of time. We need to design an auction and brokerage mechanism that uses a spot market to sell unused resources with reduced prices rather than letting them sit idle. We need to monitor cloud markets to attract customer by setting lowest possible prices for the resources.

This brokerage service could be provided by each cloud service provider individually, but would only pay off at scale. On the other hand, customers who need cloud resources to run applications without real-time constraints benefit from cloud resources ‘spot markets', particularly if many provider offers are available in the market and the cost for each of them is comparable. We will therefore target an *independent multi-cloud brokerage* model:

- Multiple providers can sell unused resources without designing and managing their own auction mechanisms, while at the same time maximize their profit without spending on monitoring and operating a resources market.
- Customers can rent cloud service with specific requirements at the lowest cost available in the market, without the need to search and compare providers and find best offers manually.

We implemented a generic architecture for a brokerage model based on tuple spaces. We will demonstrate the suitability of this architecture for cloud resource spot markets.

III. MARKET ARCHITECTURE

A. Market Implementation Principles

We have analysed architectural and functional requirements to implement sport market strategies for cloud resources [27]. Based on this, the market principles realised in the implementation of our brokerage model for cloud resources market are:

1) **Provider Resource Details**. Cloud providers using this brokerage model can sell their already unused resources, for which they set their price at offer insertion time into the cloud resource market. The resource offers need to be specified before sending the request to the broker. The offer has to contain at least these details:

   - name of the provider,
   - machine configuration (RAM in GB, CPU cores, Storage capacity in GB),
   - time that resources will be available,
   - price per hour for using these resources,
   - validation time in minutes for that offer.

2) **Customer Requirements and Matching**. Customers can customize requested resources, but there is no guarantee that available resources will exactly match their requirements. The result from the matching process is:

   - the best matching offers found from single or multiple providers for the same needed time with the same or larger resources configuration than the customer's requirements. Also, the cost will be less or the same as the customer request.

   Customers specify their requirements before sending their request to the broker service. The request has to contain:

   - minimum size of the resource configuration (RAM in GB, CPU cores, Storage in GB),
   - time needed to use the resources,
   - max. acceptable resource price per hour.

3) **Auction Mechanism**. The auction mechanism ‘*ask queue*' that is followed in this model is that the matching offers are arranged by whether they have the same or larger resources configuration than customers required and whether they increase the order in an array and start with a less requested price. Those with the same or less requested price than a customer request are auctioned off first.

4) **Market Goal**. The *goal of this model* is to find offers that match the customer request most closely and that reduce the cost of using resources as much as possible. Because of this, in some demand cases, the mechanism can combine offers from multiple providers that will reduce the cost for the customer. Furthermore, it creates the opportunity to find the required time for using the resources from multiple providers when a resource is not available from a single one. Therefore, an algorithm to compare prices and check availability times for resources is used after the step that matches the resources with customer requirements.

B. Architecture Requirements

What emerges are a number of technical requirements that a brokerage architecture needs to satisfy:

- Easy addition / removal of both providers and requesters: these can be added as agents accessing the space.
- Suitable functionality to implement offering and requesting activities: tuple space provide a communication paradigm around operations depositing, detecting and retrieving descriptive data.
- Backbone support for matching: through the associative approach, offers and requests can easily be matched.
- Scalability for the cloud: although not an intrinsic property, we can experimentally demonstrate scalability.

We have indicate the main reasons for choosing tuple spaces [30]. Other options would have included other forms of associative memory, e.g. building on query languages. Semantic matching would have been an option for the matching support, but would have had to be combined with other coordination tools. Tuple spaces provide the best coverage of the features needed – we will see that only a ranking mechanism is not directly supported (but can be seamlessly added, as we will demonstrate). We will now detail the tuple space features.

C. Tuple Space Principles

This section will overview coordination and tuple space architectures and introduce a sample Java tuple space implementation called LighTS.

A **tuple space** is an implementation of an *associative memory* paradigm, which is used to store and retrieve objects (data) and which logically works like a shared memory. The object is shared between various processes, but there is no physical memory shared. Furthermore, it can be accessed through pattern matching as an associative access form. The object in the tuple space does not belong to any process and it remains inside the tuple space until it is retrieved by some other process. There are several features of the tuple space coordination that makes it a suitable model for parallel and mobile applications. It facilitates the implementation of a cloud brokerage market to advertise services and match consumer requests. One of these features is that processes do not need to be available at the time of communication and anyone, be that provider or customer, can access the data.

The primary objects are **tuples**, which are constructs consisting of a collection of ordered lists of elements. In a tuple, elements are composed of field and value pairs:

$$[field_1,value_1; field_2,value_2; \ldots ; field_n,value_n]$$

It can have any number of field and value pairs. However, there may not be any NULL values. There are *two types of tuples*: *active* and *passive*. Passive tuples are stored in the tuple space until they are retrieved by some process, whereas tuples that can spawn other processes or perform some functions are called active. The active tuples turn into passive tuples when they have finished their task.

Templates are tuples used for matching and retrieving tuples. The format of a template is:

$$[field_1,value_1;..; field_i,NULL; .. ;field_n,value_n]$$

This format is similar to tuple format, but NULL values are allowed for specific fields *i* in the template and represent placeholders. The values that templates can have are *actual* or *formal*, or a mixture of both. In the matching process, the formal values will be replaced and the actual values will be compared. Null values would be replaced also by actual values when the template is matched against a tuple. The main difference between tuple and template is their usage. Tuples are used to insert tuples (group of data) into the tuple space, while the template is used to search for a data tuple.

The first application of tuple spaces in a programming language was *Linda*, an abstract parallel programming language used for parallel processing between objects. Linda provides four basic operations:

OUT, IN, RD, EVAL

OUT is used to store a single tuple in the tuple space; it does not return anything. The argument passed to OUT is a tuple. *IN* is used by a process to retrieve and remove a tuple from tuple space, while *RD* is used to read a tuple that is already stored in the tuple space without removing the tuple from the tuple space, it gets a copy of the tuple and returns it back to the calling process. Both IN and RD get a template as an argument. *EVAL* is used to carry out functions inside the tuple space, which turns an active tuple into a passive tuple when all the performing functions are terminated.

The Java implementation of tuple spaces is called *LighTS*, which was designed as an open-source, customizable tuple space framework. It provides Java support for basic Linda operations in a local implementation of a tuple space [6].

D. Coordination Implementation

An architecture for a multi-cloud resource market for an independent brokerage model is based on the following principles:

- The tuple space holds information relating to provider offers for the cloud resource market. Providers who want to sell their resources submit details of their offers.
- These details provided in the application are filled into the tuple fields and inserted it into the tuple space using the OUT() operation.
- Each tuple field describes one piece of information related to the resources that will be offered by the cloud providers.
- This tuple 'offer' remains in the tuple space 'spot market' until it is bought by a customer or the validation time of this 'offer' tuple has expired.

- The service removes the offer from the resource market through passing the offer details in the template fields and use the IN() operation.
- Customers submit request details, then all 'offer' tuples in the tuple space are matches using the RD() operation to find suitable matching offers.

To facilitate this, the core operations of LighTS proved not to be sufficient and needed extension. We added a MULTI-RD operation, which allows retrieving an array of matching tuples:

- MULTI-RD extracts all matching tuples for a request template. This is provided as a wrapper function on top of the LighTS implementation.

Generally, through three Linda operations (OUT, IN and RD) that LighTS supports and our extension (MULTI-RD), the cloud resource spot market can be established. The providers can sell their resources and customers can check and buy the cloud resource through the application that connects them to the cloud resource market.

E. Resource Request Matching

The matching is done after a customer submits a request to the broker service and the service reads all relevant offers are available in the spot market. The **matching algorithm** is defined as follows:

1. All offers are selected that have the same or larger resource configurations than the requirements of the customer.
2. The list of matching offers is rearranged in the ask queue in increasing order, starting with the lowest asked price by providers. In some cases (a) there may be no matching offers available for the time requested or (b) all available offers do not meet the customer's resource requirements.
3. The customer requested price and required time for resources are compared with asked price and resource availability for each matching offer in the ask queue.

At the core is a **multi-objective optimisation** problem

$$min(CPU, STRG, RAM, Time, Price)$$

based on an objective vector that defines upper and lower bounds for the objective function values of Pareto optimal solutions. The feasible set of decision vectors is defined by a constraint function on the five selection criteria above:

$$\min_{x \in X} \sum_{i=1}^{k} w_i f_i(x),$$

Linear scalarisation is here based on an equal weighting w on the five criteria f_1 to f_5 as above.

The goal of this matching algorithm is to find the lowest technically matching cost offer. The algorithm used *a pattern-matching technique* for the identification of matching requester-provider pairs. The solution relies on the *associative matching* of the tuple space. We added an intermediate step to allow for sorting to take place.

F. Composite Matches

The matching notion is extended here to *composite matches*, i.e., combining two or more provided resources to satisfy a single request [23]. The algorithm can combine two offers as a result of the matching process for the customer request if this combination reduces the cost of using the resources, possibly using resources from multiple providers as needed. The result of this matching is the most suitable low cost offer available from possibly multiple providers.

Total composite cost across all offers is the *ranking factor*. The resource requirement matching is decided as follows:

- Time required: the individual times need to add up.
- Resource configuration: the total required configuration need to be achieved as a combination of individual offers.

Our solution assumes here that the VM load to be deployed can be split as required. The algorithm iteratively starts with the closest matching offer (below the request) and tries recursively fill the gap with further offers.

G. Illustration and Application of Principles

A sample use case for both provider and customer shall illustrate the brokerage model. Firstly, a provider plans to sell their resources (e.g. VMs) in the spot market. They have to provide the information details relating to the offer and then send it to the spot market service, which will insert this offer into the cloud resource market place. An architecture for this process is presented in Figure 1.

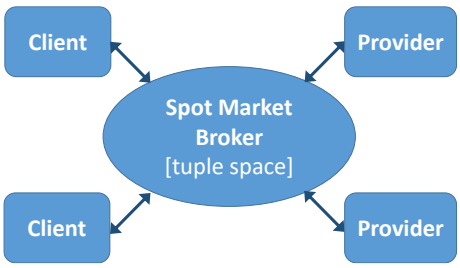

Fig. 1 Provider and Client Interaction with Broker Service

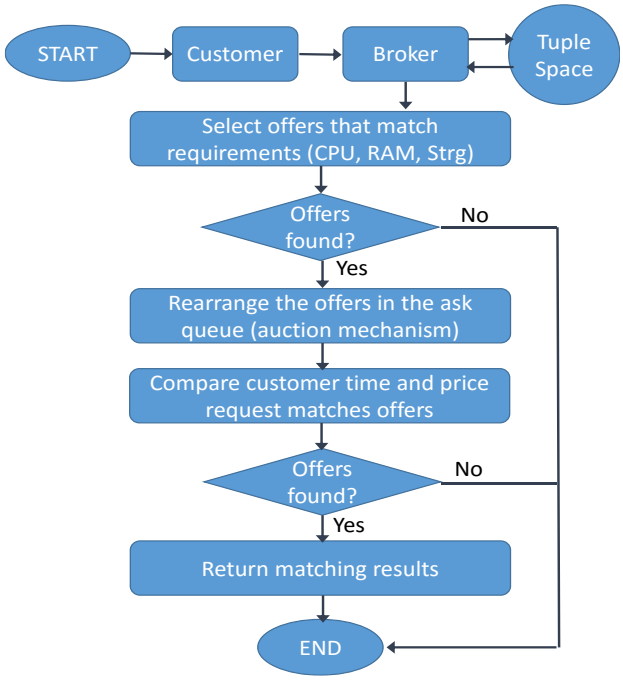

Fig. 2 Customer Submit Request Flow Process

Then, assume a customer needs to buy computing resources from spot market at a low cost. They submit their requirements to the spot market service. The service will search through the spot market, then apply the matching process to find the most suitable offer compared to the customer request. A process for this is presented in Figure 2.

Overall, the service is a broker that connects both customer and provider through the spot market. The advantage of the tuple space architecture is no need for extra coordination between spot market and broker, or between broker and providers/customers.

This implements the spot market principles presented in Section II.B. Offers are managed in terms of sorted *spot ask queues*. It considers (i) the actual cost of resources and (ii) profit maximization, specifically for the customer.

IV. Evaluation

For the evaluation of the implemented broker-based spot market service, different test scenarios were carried out to evaluate the effectiveness of the cloud resource market based on the extended tuple space architecture. The matching process is validated through the application of different request use cases and then analysing the results of the matching in relation to customer requirements.

A. Evaluation Criteria

Four concerns have been addressed to evaluate design and implementation of this market brokerage model. These criteria were considered as important points in respect of the cloud resource 'spot market':

- Firstly, the tuple space architecture for the cloud resource market is *adequate* for brokerage.
- Secondly, the auction mechanism followed in the matching (ask queue) is *effective*.
- Thirdly, the selection of best matching offers for customer requests succeeds in *reducing the cost* as much as possible.
- Finally, *scalability* is also looked at by monitoring performance for varying loads.

B. Evaluation Method

Various factors cause the actual price for cloud resources in the spot market to vary significantly: (i) the dissimilarity of the pricing policy cross the cloud providers services, (ii) the rapid change in resource price depending on the time and the day, as well as (iii) the location of the datacentre. The data used to test this model was generated randomly for this case within some set min/max boundaries, but reflects actual figures in the cloud resource market. Details for this data are displayed in Table 1.

Table 1. Validation Data.

Field	Min	Max
CPU (Cores)	1	20
RAM (GB)	1	32
Storage (GB)	50	1000
Machine Availability Time (Hour)	2	9
Price/Hour ($)	0.02	1.50
Offer Validation Time (Minutes)	10	80

In order to validate matching, different request cases were generated and submitted. Some requests were sent to the broker twice, once to search for the matching offer from single providers and the other to do the same for multiple providers. The details of a sample request are displayed in Table 2.

Table 2. First Sample Request.

Field	Value
Min. CPU (Cores)	16
Min. RAM (GB)	20
Min. Storage (GB)	800
Machine Needed Time (Hour)	8
Max. Price / Hour ($)	1.1

In all auction activities, the first step of the matching process follows the same methodology, the selection offers from the tuple space has to have the same as or a larger resources configuration as determined by the request. Then, selected offers are sorted in increasing order of price to apply the auction mechanism to the ask queue. The lowest asked price offered is more likely to be sold first. Where two selected offers have the same ask price, the offer that is available for the greatest amount of time for resources is inserted first in the ask queue. A sample request is displayed in Table 3.

Table 3. Second Sample Request.

Field	Value
Min. CPU (Cores)	20
Min. RAM (GB)	32
Min. Storage (GB)	1000
Machine Needed Time (Hhour)	7
Max. Price / Hour ($)	1.5
Search Type (Single/Multi Provider)	Multi

C. Results

This evaluation focuses on the effectiveness of the (extended) tuple space architecture for the cloud resources market brokerage model. The main role of the cloud resources market to effectively coordinate participants (both buyer and seller) is confirmed through the results related to the data that has been generated and tested in this model – see below. The application connects easily and interacts directly through our extended tuple space architecture.

Architecture. The architecture of the cloud resource market is based on the tuple space architecture. The spot market, which is the place that resources are sold at the price set by the market or seller (cloud service providers here), is implemented using coordination actions. The flexibility of this architecture, by coordinating any number of tuples and accessing processes in the tuple space, allows cloud providers to insert offers when they wish to sell unused resources, to determine the price they would like to sell their resources at and the duration that the offer is valid for to buy, before it is removed from the spot market. Customers can view offers at any time and can submit requests to find best matching offers. Offer details, represented by a tuple, can by be edited (e.g., increased) or removed easily by associative access. The coordination application acts as a broker between provider and customer. This makes the *architecture adequate* supporting the market functions.

Auction Mechanism. Various generated requests were tested in this model, including for instance when the resources configuration that the customer requested was too high (cf. Table 3). The process laid out in Section 4.2 was tested with the test data, confirming the *auction mechanism* is *effective*.

Cost Optimisation. The solution is effective in terms of optimising the outcome. For the first request, the result of the first step in the matching process was 15 matching offers from 54 offers available in the tuple space for both cases. Then the suggested result after the comparison was applied, which was different in both cases. In the first case, the matching offer was from a single provider with a price approximately equal to the required price, while the second case was from two providers with a cost less than required (0.78$). In addition, if the same previous request case entered offers after 30 minutes and 60 minutes, the number of matching offers changes depending on the number of offers that are available at that time. The service updates the tuple space every few minutes to remove expired offers from the spot market.

The matching offer details for the two sample cases are described in Tables 4 and 5. Our linear optimisation solution is in line with other cost optimisation solutions and linear programming approaches, such as [19].

Table 4. Result for First Case.

Prov	CPU	RAM	Strg	Val Time	Avail Time	Price/Hour
P44	16	20	1000	80	8	1.08

Table 5. Result for Second Case.

Prov	CPU	RAM	Strg	Val Time	Avail Time	Price/Hour
P40	16	20	800	40	6	1.00
P41	16	20	800	60	2	1.01

In the second request, the results for the first step in the matching process are just 3 matching offers. Then the result of the best matching offers for this request that was generated by the matching process is from different providers. This combination of offers can reduce the total cost of using resources. In this generated case for example, the cost will reduce from 10.5 $ to 9.65$ for the 7 hours requested. The suggested offers are described in Table 6. Thus, we conclude that the solution does *optimise the costs*.

Table 6. Ranking of Suggested Offers.

Prov	CPU	RAM	Strg	Val Time	Avail Time	Price/Hour
P52	20	32	1000	40	3	1.35
P53	20	32	1000	60	5	1.40

Scalability. Scalability and other aspects were already addressed elsewhere [1,5]. We varied between tuple sizes to cover 100, 1000 and 10000 bytes. Tests were run with 100, 1000 and 10000 requesters and providers, then doubling and tripling their respective numbers separately to deal with imbalances. Each test was run 10 times in a networked environment. The results *demonstrate acceptable performance* even for very high loads (although the increase is not linear).

As noted early, we opted for a linear scalarisation approach that performs comparably with other approaches [19], but a greater concern in the market setting is scalability for larger numbers of participants, which is less well experimentally demonstrated. Here we can demonstrate success.

D. Discussion and Threats to Validity

We can conclude is the cloud resource market can be operated based on a tuple space architecture. In addition, the service that was implemented in this spot market model works in the same way as any online broker services that connect companies and customers through listing the company's products to the customers and indicating them the best product that matches their requirements. The tuple space acts as the datacentre to store all relevant information.

However, the architecture of this model can be enhanced by adding more brokers between buyers and customers or between them and allowing the spot market to provide more services in the cloud resources market. Moreover, the matching process can be extended in both steps easily to increase the details included in the resources request and also to specify more restrictions in the selling template of the resources as per the cloud providers' wishes.

We present a solution that focuses on the customer as the beneficiary. This can be seen as a limitation, as the providers' needs to optimise their revenue is ignored. However, current single provider solutions ignore the customer, and in a scenario of a broker provided by a third party, better customer prospects need to be considered. However, further investigations into how to jointly facilitate customers and providers are required.

V. Related Work

We have already reviewed related work on cloud resource markets. Here, we only look at architectural aspects. For instance, Mong Sim covers what is done at each resource provider for VM spot instances provisioning [25].

There is work using a tuple space architecture in the cloud as a coordination model for cloud service and also as a service, which matches the requests with providers. Some of these prove the scalability of the architecture being able to perform well with significantly large numbers of providers and requesters [9,5].

In addition, there is research that proposes different approaches to finding the best cloud providers for the customers as requested. The aim in the most of these is to perform the matching between customers' request and cloud providers automatically [14]. Such spot markets are being operated by some cloud service providers.

Some investigate the rules of market and auction mechanisms that have different methodologies and architecture design from one provider to another. Auction mechanisms have been implemented [20] as a market place for computing resources, using for instance a queuing approach. A recent commercial solution is Deutsche Börse Cloud Exchange (https://cloud.exchange), though this venture signals the teething problems of an early stage of acceptance, here not promoting business value for users well enough.

VI. Conclusions

In this paper, a brokerage model for a cloud resources spot market has been discussed in terms of market principles and the architecture to operate this spot market. The matching

process between customer request and offers from the cloud resources provider was examined and discussed. The evaluation showed that the suggested architecture, based on an extended tuple space, with this brokerage model works effectively for a cloud resources market. In addition, the performance of the matching process that uses the request queue technique in the auction mechanism and combines multiple cloud resource providers, increases the chance of matching between demand and supply and decreases costs of using resources for the customer, if this process works properly in the matching requests with cloud providers as illustrated.

Please note that our objective here was not to provide a fully optimal brokerage solution (although our composite matches is non-trivial), but to demonstrate the suitability of the tuple space architecture to provide a multi-cloud broker solution for cloud resource spot markets that goes beyond current single-provider solutions. We expect multi-cloud brokerage to play a more significant role in the future [18,24], beyond the current single-provider markets offered by some. We expect these to emerge similar to last minute brokering services in non-technical domains. Marketplace for cloud resources, particularly at the SaaS layer have already appeared.

For future work, the architecture of this independent brokerage model can be extended to allow for resources to be automatically allocated to the customer instead of transferring the customer to the cloud providers for service deployment [26]. Also, the resource information schema can be extended towards an ontology [4,21] offers to improve the selection of the offers in the first step of the matching process, e.g. reliability or reputation. Matching can be improved to find the best providers from multiple aspects, not just considering price and availability [15]. Another direction is to use containers [31] for implementation.

ACKNOWLEDGMENTS

This work has received funding from IC4 (an Irish National Cloud Computing Technology Centre funded by EI and the IDA).

REFERENCES

[1] Hari, H., "Tuple Space in the Cloud," Master's Thesis. University of Uppsala. 2012.

[2] Fowley, F., Pahl, C., and Zhang, L., "A comparison framework and review of service brokerage solutions for cloud architectures," International Conference on Service-Oriented Computing ICSOC'2013 Workshops, pp. 137-149. 2014.

[3] Sharma, B., Thulasirm, R., Thulasirman, P. and Grag, S., "Pricing Cloud Compute Commodities: A Novel Financial Economic Model," IEEE/ACM Intl Symp on Cluster, Cloud and Grid Computing (CCGrid). 2012.

[4] Pahl, C., "An ontology for software component matching," International Journal on Software Tools for Technology Transfer, 9(2). pp. 169-178. 2007.

[5] Creaner, G. and Pahl, C., "Flexible Coordination Techniques for Dynamic Cloud Service Collaboration," Adaptive Web Services for Modular and Reusable Software Development - Tactics and Solutions. 2007.

[6] Picco, G.P., "LighTS," Available from: http://lights.sourceforge.net/ [Accessed 25 April 2016]. 2001.

[7] Pal, R. and Hui, P., "Economic Models for Cloud Service Market (Pricing and Capacity Planning)," Telekom Innovation Laboratories. 2015.

[8] Son, J., "Automated Decision System for Efficient Resource Selection and Allocation in Inter-Clouds," Department of Computing and Information System. The University of Melbourne. 2013.

[9] Pahl, C. and Xiong, H., "Migration to PaaS Clouds - Migration Process and Architectural Concerns," International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems MESOCA'13, 2013.

[10] Abhishek, V. Kash, I. and Key, P. "Fixed and Market Pricing for Cloud Services," NetEcon. 2012.

[11] Zafer, M., Song, Y. and Lee, K., "Optimal Bids for Spot VMs in a Cloud for Deadline Constrained Jobs," IEEE Intl Conf on Cloud Computing. 2012.

[12] Toosi, A., Van Mechelen, K., and Buyya, R., "An Auction Mechanism for a Cloud Spot Market," Cloud Computing and Distributed Systems Laboratory. 2014.

[13] Al Roomi, M., Al Ebrahim, S., Buqrais, S., and Ahmad, I., "Cloud Computing Pricing Models: A Survey, " Intl Jrnl of Grid and Distr Computing 6(5), pp.93-106. 2013.

[14] Gilia, P. and Sood, S., 2013. "Automatic Selection and Ranking of Cloud Providers using Service Level Agreements," Intl Journal of Computer Applications, 72(11). 2013.

[15] Redl, C., Breskovic, I., Brandic, I., and Dustdar, S. 2012. "Automatic SLA Matching and Provider Selection in Grid and Cloud Computing Markets," In ACM/IEEE Intl Conference on Grid Computing (GRID '12). 2012.

[16] Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., and Tsafrir, D., 2011. "Deconstructing Amazon EC2 Spot Instance Pricing," In IEEE Third Intl Conference on Cloud Computing Technology and Science. 2011.

[17] Shang, R., Huang, J., Yang, Y., and Kauffman, R.J., "Analyzing the Impact of Cloud Services Brokers on Cloud Computing Markets," Pacific Asia Conference on Information Systems. 2013.

[18] Fowley, F., Pahl, C., Jamshidi, P., Fang, D., and Liu, X., "A Classification and Comparison Framework for Cloud Service Brokerage Architectures," IEEE Transactions on Cloud Computing. 2016.

[19] Bessa, R.J. and Matos, M.A., "Optimization models for EV aggregator participation in a manual reserve market," Power Systems, IEEE Transactions on, 28(3), 3085-3095. 2013.

[20] Henzinger, T.A., Singh, A.V., Singh, V., Wies, T. and Zufferey, D., "A marketplace for cloud resources," In ACM international conference on Embedded software (EMSOFT '10). 2010.

[21] Greenwell, R., Liu, X., Chalmers, K., and Pahl, C., "A Task Orientated Requirements Ontology for Cloud Computing Services," 6th Intl Conf on Cloud Computing and Services Science Closer. 2016.

[22] Maamar, Z., Dorion, E., and Daigle, C., "Toward Virtual Marketplaces for E-Commerce Support," Commun. ACM 44(12): 35-38. 2001.

[23] Wang, M.X., Bandara, K.Y., and Pahl, C., "Integrated constraint violation handling for dynamic service composition," IEEE International Conference on Services Computing SCC'09, 2009.

[24] Li, Z., Tärneberg, W., Kihl, M., and Robertsson, A., "Using a Predator-Prey Model to Explain Variations of Cloud Spot Price," 6th Intl Conf on Cloud Computing and Services Science Closer. 2016.

[25] Mong Sim, K., "A Price and Time Slot Negotiation Mechanism for Cloud Service Reservations," IEEE Transactions on Systems, Man, and Cybernetics, vol. 42, no. 3, 2012.

[26] Jamshidi, P., Ghafari, M., Ahmad, A., and Pahl, C., "A framework for classifying and comparing architecture-centric software evolution research," European Conference on Software Maintenance and Reengineering, 2013.

[27] Pahl, C. and Giesecke, S. and Hasselbring, W., "Ontology-based modelling of architectural styles," Information and Software Technology, 1 (12). pp. 1739-1749. 2009.

[28] Comuzzi, M. and Pernici, B., "An architecture for flexible web service QoS negotiation". In EDOC Conference, pp. 70-82. IEEE, 2005.

[29] Prodan, R., Wieczorek, M., and Fard, H., "Double Auction-based Scheduling of Scientific Applications in Distributed Grid and Cloud Environments," Journal of Grid Computing, Vol. 9, pp. 531-548, 2011.

[30] Doberkat, E.-E., Franke, W., Gutenbeil, U., Hasselbring, W., Lammers, U., and Pahl, C., 1992. "PROSET - a Language for Prototyping with Sets," International Workshop on Rapid System Prototyping, pp. 235-248. 1992.

[31] Pahl, C., "Containerisation and the PaaS Cloud, " IEEE Cloud Computing, 2(3). pp. 24-31, 2015.