Multi-agent Virtual Machine Management Using the Lightweight Coordination Calculus

Multi-agent Virtual Machine Management Using the Lightweight Coordination Calculus

Paul Anderson, Shahriar Bijani, and Herry Herry

School of Informatics, University of Edinburgh,
10 Crichton Street, Edinburgh, EH8 9AB, UK
{dcspaul,s.bijani}@ed.ac.uk
h.herry@sms.ed.ac.uk

Abstract. LCC is a Lightweight Coordination Calculus which can be used to provide an executable, declarative specification of an agent interaction model. In this paper, we describe an LCC-based system for specifying the migration behaviour of virtual machines within, and between datacentres. We present some example models, showing how they can be used to implement different policies for the machine allocation and migration. We then show how LCC models can be used to manage the workflows that involve creation and deletion of virtual machines when migrating services between different datacentres.

Keywords: autonomic computing, multi-agent systems, virtual machines, OpenKnowledge, Lightweight Coordination Calculus

1 Introduction

Virtualisation technology has recently transformed the availability and management of compute resources. Each *physical machine* (PM) in a datacentre is capable of hosting several *virtual machines* (VMs). From the user's point of view, a virtual machine is functionally equivalent to a dedicated physical machine; however, new VMs can be provisioned and decommissioned rapidly without changes to the hardware. VMs can also be *migrated* between physical machines without noticeable interruption to the running applications. This allows dynamic load balancing of the datacentre, and high availability through the migration of VMs off failed machines. The resulting *virtual infrastructure* provides the basis for *cloud computing*.

Managing the placement and migration of VMs in a datacentre is a significant challenge; existing commercial tools are typically based on a central management service which collates performance information from all of the VMs. If the current allocation is unsatisfactory (according to some policies), then the management service will compute a new VM allocation and direct agents on the physical machines to perform the necessary migrations.

As the size and complexity of datacentres increases, this centralised management model appears less attractive; even with a high-availability management

service, there is possibility of failure and loading problems. If we would like to extend the domain of the virtual infrastructure to encompass multiple datacentres, managed by different providers, then the central model is no longer appropriate; in this federated "cloud" scenario, there may no longer be a single organisation with ultimate authority over all of the infrastructure.

This motivates us to propose a less centralised solution where agents located on the physical machines negotiate to transfer VMs between themselves, without reference to any centralised authority. This seems particularly appropriate for many situations where a globally optimal solution is not necessary or feasible; for example, if a machine is overloaded, it is often sufficient to find some other machine which will take some of the load. Likewise, an underloaded machine simply needs to take on additional VMs to improve its utilisation; there is no need for any global knowledge or central control.

However, moving virtual machines *between* datacentres is also more difficult: in general it is not possible to perform a live migration, and a new virtual machine must be started in the target datacentre, and the services transferred, before stopping the original virtual machine. The new machine will also have a different IP address, and possibly other differences, which mean that the migration may not be transparent to clients of the service. In this case, the clients will need to be notified about the change, and a comparatively complex workflow may be needed to avoid any break in the service. Once again, there may be no obvious central authority to sequence this workflow, and this motivates an agent-based approach to the workflow execution.

In this paper, we present a solution to the above problem where agents follow *interaction models* (IMs) described in the *lightweight coordination calculus* (LCC). The agents use the OpenKnowledge framework to locate appropriate interaction models and to identify suitable peers. These interaction models specify the agent behaviour, and allow them to make autonomous decisions; for example, the choice of VM to accept could be based on local capabilities, the properties of the VM being offered, the financial relationship with the donor, etc. Once a transfer has been agreed, the participating machines will execute interaction models which implement a workflow to effect the transfer. This may be a simple live migration which is transparent to any clients of the service, or it may be a more complex workflow which involves notifying clients, and stopping and starting virtual machines in different datacentres.

One important consequence of this approach is that we can very easily change the global policy of an entire infrastructure by introducing new interaction models. For example, a particular model may encourage the physical machines to distribute the load evenly among themselves; this makes a lightly-loaded infrastructure very agile and able to accept new VMs very quickly. Alternately, a different interaction model may encourage the machines to prefer a full, or empty, loading as opposed to a partial one. Some of the machines would then be able to dispose of all their VMs, allowing them to be turned off and hence saving power.

Section 2 provides some background on LCC and the OpenKnowledge framework, and section 3 presents LCC interaction models for various scenarios involving virtual machine allocation. These interaction models, together with a live prototype which implements them on a real datacentre, are described in more detail in [1]. Section 4 presents new work which describes an extension of the interaction models to manage the workflows which are necessary to deploy the allocations when services are migrated between datacentres and live migration is not possible. Section 5 covers some of the consequences and issues raised by this approach, and section 6 provides a brief discussion of some related work on VM management, including the state-of-the-art in commercial tools, as well as more experimental, agent-based approaches.

2 LCC and OpenKnowledge

A computational agent - such as one responsible for one of our physical machines - must be capable of acting autonomously, but it will also need to communicate with other agents in order to achieve its goals. In a multi-agent system (MAS), the agents often observe conventions which allow them to co-operate. These are analogous to the *social norms* in human interactions, and may be more or less formal – an oft-cited example is the rules which govern the bidding process in an auction. In our application, agents must be able to compare the respective resource utilisation of their hosts, and reach an agreement about the transfer of a virtual machine. Typically, the social norms in a MAS will be defined using an explicit protocol. The *lightweight coordination calculus* (LCC) is a declarative, executable specification language for such a protocol.

2.1 LCC

LCC [3] is based on a process algebra which supports formal verification of the interaction models. In contrast with traditional specifications for *electronic institutions*, there is no requirement to predefine a "global" script which all agents follow - the protocols can be exchanged and evolved dynamically during the conversation. LCC is used to specify "if" and "when" agents communicate; it does not define how the communication takes place[1], and it does not define how the agents rationalise internally. There are several different implementations of the LCC specification, including OpenKnowledge (see below), Li[2], UnrealLCC[3] and Okeilidh[4].

There is insufficient space here to describe the LCC language in detail; the OpenKnowledge website contains a good introduction[5], and there are also some

[1] The inter-agent communication mechanism is defined by the implementation.
[2] `http://sourceforge.net/projects/lij`
[3] `http://sourceforge.net/projects/unreallcc`
[4] `http://groups.inf.ed.ac.uk/OK/drupal/okeilidh`
[5] `http://groups.inf.ed.ac.uk/OK/index.php?page=tutorial.txt`

video tutorials[6]. The following brief summary should be sufficient to follow the annotated example presented in the next section:

Each IM includes one or more clauses, each of which defines a *role*. Each role definition specifies all of the information needed to perform that role. The definition of a role starts with: `a(`*roleName, PeerID)*. The principal operators are outgoing message (`=>`), incoming message (`<=`), conditional (`<-`), sequence (`then`) and committed choice (`or`). Constants start with lower case characters and variables (which are local to a clause) start with upper case characters. LCC terms are similar to Prolog terms, including support for list expressions. Matching of input/output messages is achieved by structure matching, as in Prolog.

The right-hand side of a conditional statement is a *constraint*. Constraints provide the interface between the IM and the internal state of the agent. These would typically be implemented as a Java *component* which may be private to the peer, or a shared component registered with a discovery service. One advantage of the separation of interaction models from the constraints is that the interaction models can easily be shared.

2.2 OpenKnowledge

OpenKnowledge (OK[7])[4,5] provides an implementation of LCC, together with some additional functionality, including a distributed *discovery service* (2.2) and an *ontology matching service* (2.2). Having decided to participate in a particular interaction, peers register their desired roles with the discovery service. This identifies a suitable set of peers to fulfil each role in the interaction. The peers are then notified and the interaction proceeds without further involvement of the discovery service[8].

The Discovery Service: In addition to locating peers with matching roles, the OK discovery service provides facilities for discovering and distributing both interaction models and components (OKCs). This means that a physical machine (in our application) need only register its willingness to participate, and the behaviour will then be defined by the IMs and OKCs which are retrieved from the discovery service. Each peer has a choice of interaction models to suit various different scenarios, but once it has subscribed to an IM, all of the peers in that interaction will be following the same "script".

The OK implementation is a scalable, open, efficient and robust service based on top of the FreePastry DHT implementation. This relies on keyword matching and is based-on a completely decentralised storing mechanism that requires

[6] `http://stadium.open.ac.uk/stadia/preview.php?whichevent=984&s=29`

[7] `http://groups.inf.ed.ac.uk/OK/`

[8] In practice, the OK implementation elects a random peer to be a coordinator for the interaction, and the coordinator executes the IM, only making calls to other peers when it is necessary to evaluate a constraint. However, this is largely an optimisation decision and different implementations take different approaches.

O(log(N)) messages to store and search for N peers (see [6] for a discussion of the implementation). In a large scale evaluation, the OK discovery service significantly outperformed the two reference approaches [7].

Ontology Matching: A major strength of the OK system is that there is no need for a *global* agreement on interaction protocols. Any group of peers can subscribe to an IM which may be publicly available, or shared between a restricted group using some private mechanism. Likewise, there is no need for a global agreement on vocabulary for the OKCs or roles – there only needs to be agreement between those peers participating in a particular interaction, and only on those terms which appear in that interaction. Rather than an a-priori semantic agreement amongst component designers (which does not scale), the OpenKnowledge implementation provides dynamic ontology coordination at runtime. This uses various different mechanisms such as structural semantic matching and statistical analysis.

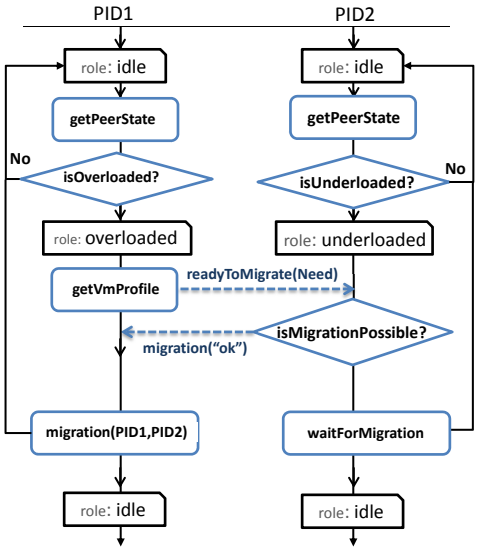

Fig. 1. The interaction diagram of a live migration: overloaded peer PID1 and underloaded peer PID2 interact to balance their loads.

3 Interaction Models for VM Allocation

In this section we describe interaction models for managing the negotiation and transfer of virtual machines between two physical machines in the same data-centre (i.e. where live migration is possible). In the first instance we implement a simple policy which aims to migrate VMs from busy peers to underloaded peers in order to balance the load of each peer.

There are three states: *idle, overloaded* and *underloaded*. The *idle* state is the initial and the goal state, in which the peer is balanced. Each peer is assumed to be balanced at the beginning of the interaction. It may then change state based on its load, or other factors[9]. Once a peer becomes unbalanced, it advertises its status to the discovery service where it will be matched with *potential* candidates for a transfer. The peer then negotiates with these candidates to find one which is prepared to participate in the transfer. The conditions for acceptance of the transfer, and the complexity of the negotiation are completely determined by the interaction models of the individual peers – these may depend on, for example, security policies or cost considerations as well as the capabilities of the physical machine (processing power, network bandwidth, memory, etc.).

Figure 1 shows the interaction diagram of a very simply implementation[10], and figure 2 shows the corresponding LCC code. After an exchange of VMs, both agents revert to the "idle" role. If they are balanced, no further action takes place. Otherwise the unbalanced peers query the discovery service again for more potential exchange partners.

The feasibility of this model for a real live system has been validated using a prototype implementation based on a real physical cluster. This is described in more detail in [1]. In addition, we used a simple simulator to investigate the behaviour of more complex models with a larger number of machines and more controlled loading. Figure 3 shows the results of this interaction model applied to 50 simulated virtual machines running on 15 physical machines. In this example, physical machines offload VMs if they have a load greater than 120% of the average, and they accept VMs if they have a load less than 80%. Initially, the VMs are allocated randomly and the resulting load is uneven. The system stabilises after a time with all the physical machines except one within the desired range (the load on the remaining machine cannot be reduced because all of the machines have a load greater than 80%). Further results and details of the simulator are available in [8].

It may be the case that we would prefer to have the minimum number of active peers, each using almost all of their resources (e.g. to minimise the cost). A major advantage of the proposed approach is that such changes to the overall policy can be easily implemented by deploying a new LCC specification which implements a different interaction model. An implementation of this alternative

[9] For example, a peer which needs to be taken down for maintenance needs simply declare itself to be "overloaded" in order to dispose of all its virtual machines.

[10] Single-corner rectangles, diamonds and dashed-arrows represent agent roles, constraints, and message passing between agents, respectively.

```
1  // Definition of the "idle" role. Here, "idle" means the "balanced" state
2  a(idle, PeerID) ::
3    // the constraint to check the state of the peer
4    null <- getPeerState(Status) then
5    //select the next state based on the peer's status
6    (
7      // if the peer is overloaded, change its role to "overloaded" and pass the status
8      a(overloaded(Status), PeerID)<- isOverLoaded() then
9    ) or (
10     // if the peer is underloaded, change its role to "underloaded"
11     a(underloaded(Status), PeerID) <- isUnderLoaded() then
12   ) or
13   // otherwise, remain in the idle role (recursion)
14   a(idle, PeerID)
15
16 // Definition of the "overloaded" role. "Need" is the amount of resources required
17 a(overloaded(Need), PID1) ::
18   // send the "readyToMigrate(Need)" message to an underloaded peer
19   readyToMigrate(Need) => a(underloaded, PID2) then
20   // wait to receive "migration(ok)" from the underloaded peer
21   migration("ok") <= a(underloaded, PID2) then
22   // live migration: send VMs from this peer to the underloaded peer
23   null <- migration(PID1, PID2) then
24   // change the peer's role to "idle"
25   a(idle, PID1)
26
27 // Definition of the "underloaded" role: "Capacity" is the amount of free resources
28 a(underloaded(Capacity), PID2) ::
29   // receive the "readyToMigrate(Need)" message from an overloaded peer
30   readyToMigrate(Need)<= a(overloaded, PID1) then
31   // send back the "migration(ok)" message, if the migration is possible, e.g.
32   // free "Capacity" of this peer > "Need" of the overloader peer
33   migration("ok") => a(overloaded, PID1) <-
         isMigrationPossible(Capacity, Need) then
34   null <- waitForMigration() then
35   // change the peer's role to "idle"
36   a(idle, PID1)
```

Fig. 2. An LCC implementation of the interaction in figure 1.

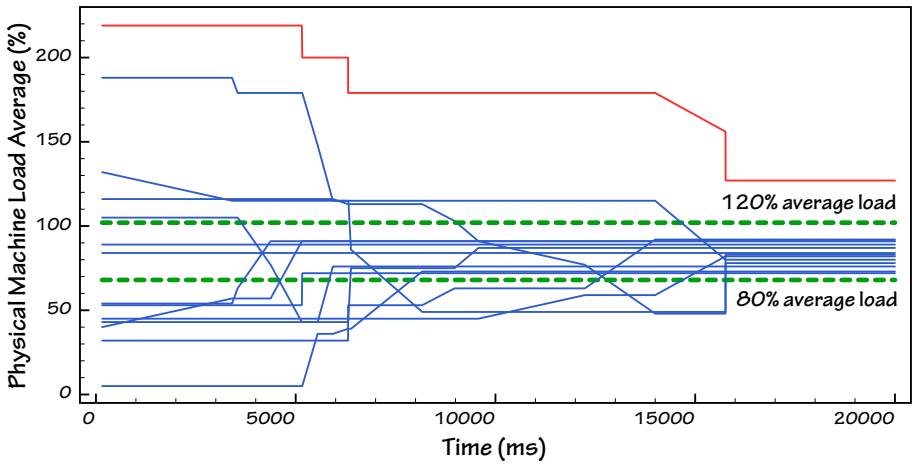

Fig. 3. A simulation showing the load on 15 physical machines as they interact to balance a load of 50 virtual machines.

policy requires only a very small change to the LCC code, and is shown in full in [1][11].

In general, the LCC will provide a clear description of the interaction, and the constraints (usually implemented in Java) will be used to implement the interface to the machine (hypervisor) itself – for example detecting the load status. The policy itself may be defined either in the lcc, or within the constraints, or in a combination of both. Since the Java components, and the LCC interaction models can both be retrieved from the discovery service, the choice here depends on which is most appropriate in each case.

4 Managing the Workflow

Having negotiated to transfer a VM *within* the same datacentre , live migration is transparent and effected with a simple instruction to the hypervisor – the VMs and their clients need not be aware that the transfer has occurred. However, if the transfer is to occur *between* datacentres, then an *offline-migration* will be required and there will be a number of changes which are not transparent to any clients of the service – in particular, the IP address of the service will usually change. To maintain service during such a transfer requires a careful sequence of operations. Traditionally, this would be orchestrated by a centralised workflow engine (see section 6). However, in a distributed, federated environment this approach to the workflow suffers from exactly the same problems as a centralised

[11] All of the LCC code for the models described in this paper is also available from `http://homepages.inf.ed.ac.uk/dcspaul/publications/ijicic.lcc`

approach to the allocation. In this section, we describe a typical pattern which occurs during service transfer, and we show how LCC interaction models can be used to sequence the necessary workflow without the need for a central controller.

In a very typical situation for a "cloud" environment, a particular service may need to be migrated from one datacentre to another – perhaps this is necessary because the internal capacity has been reached, or because of failures, or for contractual reasons. If the service has active clients, then the new copy of the service must be started, and all of the clients then transferred, before the original service can be stopped. This sequence is illustrated in figures 4 to 7.

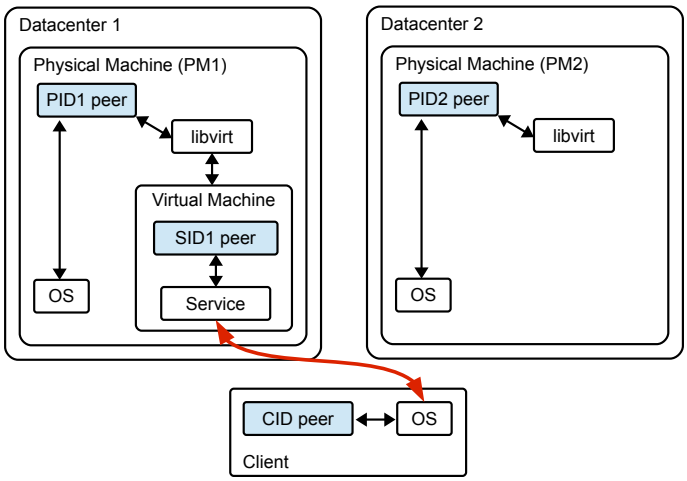

Fig. 4. The initial state of the system: A client is using a service which is running in datacentre 1.

This workflow can be implemented in a fully distributed way using LCC. The workflow is separated into a number of roles which are assigned to associated agents, and the interaction between these agents will automatically execute the workflow. Figure 8 shows the corresponding interaction diagram[12].

This interaction operates as follows:

1. **Initial State** (figure 4)
   - The initial state includes a client which is using a service provided by a virtual server in datacentre 1. The client is managed by an agent (CID) which is capable of redirecting the reference from one service to another.
   - The service itself is managed by an agent (SID1), and is running on a virtual machine (VM1). This is hosted on a physical machine managed by agent (PID1).

[12] The full LCC code is available at `http://homepages.inf.ed.ac.uk/dcspaul/publications/ijicic.lcc`

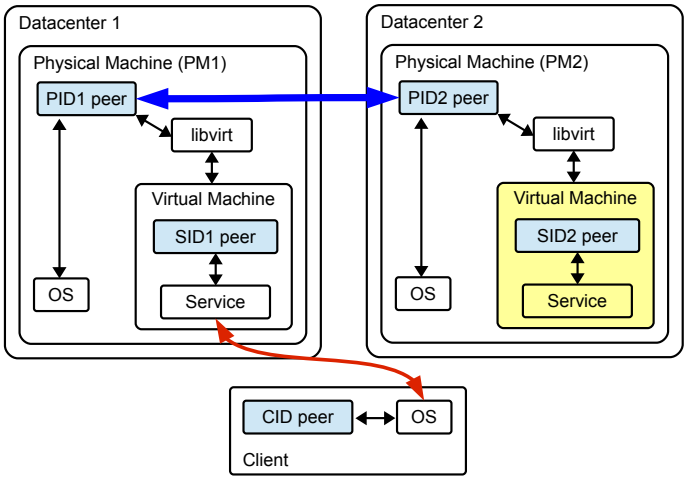

Fig. 5. The first stage of the workflow: A transfer of the service to datacentre 2 has been agreed, and a new virtual machine is started in the target datacentre to host the service.

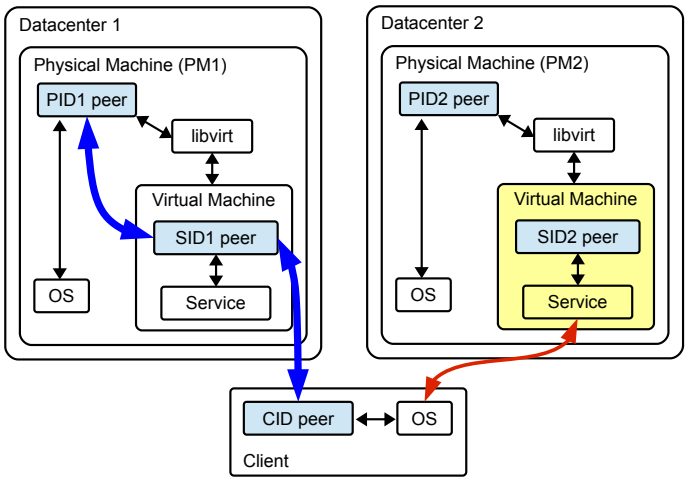

Fig. 6. The second stage of the workflow: The client is notified about the imminent removal of the original service and it locates and reattaches to the new service.

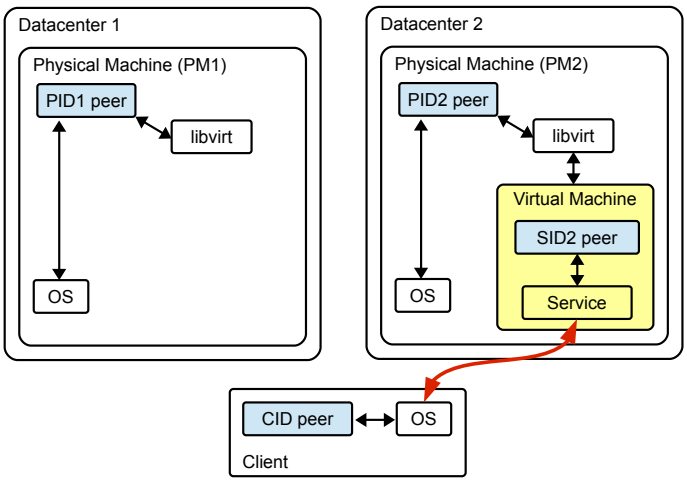

Fig. 7. When the client has left the original service, the service shuts down and the virtual machine is deleted.

- A similar agent (PID2) is managing another physical machine (PM2) in datacentre 2.
- The agents for all of the physical machines start in the "initial" role.

2. **Stage 1** (figure 5)
   - Now assume that PM2 has spare capacity and it therefore moves from the "initial" role into the state "canAcceptLoad". As in the example from the previous section, this is now registered with the discovery service as available to negotiate the acceptance of additional VMs.
   - If PM1 now needs to be removed from service for some reason, PID1 must migrate all of its virtual machines to another physical machine. It therefore moves into the "emigrant" role and is matched with PID2 by the discovery service.
   - Once the service transfer has been agreed, PID2 starts a new virtual machine (VM2) to host the new service instance, and informs PID1 when the service is available.
3. **Stage 2** (figure 6)
   - Before deleting the VM from PM1, agent PID1 must contact all of the clients of the service and inform them that the service is shutting down, and that they should redirect.
   - The client agents will attempt to locate a replacement server using the discovery service, and will reattach to the newly started service on VM2 (or possibly, some other alternative service).
4. **Final State** (figure 7)
   - Once all of the clients have redirected away from the original service, the VM on PM1 can be deleted.
   - PM1 is now free from virtual machines and able to shut down.

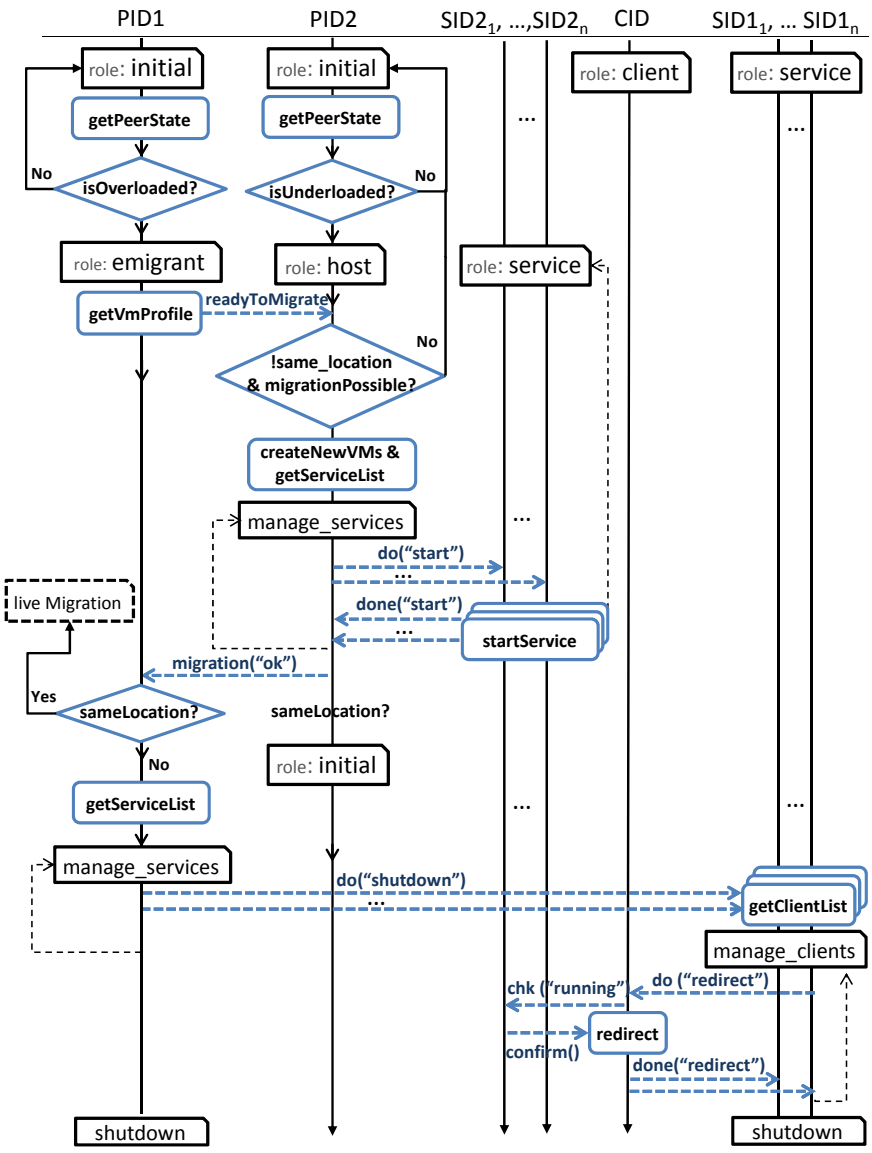

Fig. 8. An interaction diagram for an LCC interaction model of offline migration.

5 Discussion and Evaluation

5.1 Centralised vs Distributed Approaches

The examples provided above have been designed to explore an extreme version of the distributed approach to the VM migration problem – individual peers have no overall knowledge of the system, and interactions are *choreographed* by a small number of peers interacting among themselves. This is a deliberate contrast to the conventional approach where a single controller with global knowledge *orchestrates* the entire interaction. In practice however, there is a continuous spectrum between these approaches - even the fully centralised tools devolve some details of the migration process to protocols which operate directly between the peers. And our distributed version relies on a discovery service which could be viewed as a type of centralised service.

The relative advantages and disadvantages of these approaches will vary, depending on the desired policies. For example, attempting to balance the load exactly across a whole datacentre clearly requires knowledge and comparison of the load on every machine. In this case, the distributed solution has no benefits and the extra overhead means that it will not perform as well as a simple centralised service. If however, we have a very large number of potential machines, and we only need (for example) to negotiate with *any* machine having a particular property, then the distributed solution excels by avoiding the (performance and reliability) bottleneck of a central controller.

However, one of the key strengths of the proposed approach is the flexibility with which the entire policy can be changed – it is easy to imagine a policy in which one machine assumes the role of "controller" and proceeds to orchestrate the remaining machines (or some subset of them) in a conventional way, thus emulating a centralised solution. So, functionally, the centralised approach is subsumed under our more general approach. Trecarichi et al. [9] showed that the OK system can support both centralised and decentralised architectures in this way. Their experimental results in an emergency response application, demonstrate similar outcomes and comparable performance under the ideal assumptions for both cases.

For any given situation, an interaction model can be chosen which operates at a specific point on the centralised/decentralised spectrum to suit the current requirements – for example, we may choose more or less complex negotiations which yield a more or less efficient solution. These interaction models may be even be run simultaneously (between disjoint sets of peers). In practice, we might expect to see a hierarchical model evolve, based on geographical and organisational boundaries – perhaps with tightly coupled protocols achieving high efficiency among the local machines, and more flexible protocols negotiating remote transfers based on more complex factors such as latency and cost.

In a real, production system there would also be a range of non-functional requirements to consider. Security, for example, may be considered simpler to guarantee in the centralised case where there is a single point of authority. Alternatively, the distributed model with restricted capabilities for individual agents

may have security advantages (see [10] for a discussion of security attacks and proposed solutions in an LCC-based system). Similarly, a centralised system presents a single point of failure which would seem to be inherently less reliable than a more distributed system. However, we have not explored these issues in detail – they will depend heavily on the details of the implementation, and it would not be meaningful to compare our prototype to a highly-engineered production system. There are also many different approaches which could be taken to the implementation of an LCC-based system – OpenKnowledge and Okeilidh for example use completely different discovery services (Pastry vs OKBook), take different approaches to the coordination of the interactions (elected coordinator vs fully distributed messaging), and use different underlying protocols.

5.2 Policies and Complexity

One non-functional requirement which merits further discussion is the ease with which the interaction models can be created, understood, tested, and validated. LCC specifications have the advantage of a small syntax which is both lightweight and simple – this is easily understood both for designing new policies or modifying existing ones. However, operators of large data centres are unlikely to cede control of their resources to systems which may exhibit unexpected emergent behaviour, or unpredictable policy conflicts. Such problems are clearly possible, but they are mitigated by several factors:

- Only the "owner" of a machine may control which interaction models that peer is permitted to subscribe to.
- Within any particular interaction, all of the participating peers are following the same interaction model. This means that all of the participants will share a common goal, and a consistent policy.
- These interaction models can be model-checked to verify properties of their behaviour (see, for example [11] where the interaction models are translated into the μ-calculus for verification).
- The LCC language makes the interaction model very explicit, and supports tools for analysis and visualisation of the interactions. This is preferable to having interactions embedded implicitly in the implementation code.

Within an individual interaction, conflicts are unlikely to occur – all of the peers will be following the same (potentially formally verified) interaction model and they will have voluntarily subscribed to this model trusting both the IM author, and the other peers. Of course, it is possible to envisage several problem scenarios: certain peers may not follow their claimed role (due to error or malicious intent), peers may adopt a policy for a new interaction which negates the previous one, etc. However, these issues are no more problematic than in a conventional centralised solution, and indeed, the explicitness and isolation of the policies is likely to make such problems easier to detect and rectify.

5.3 Peer and IM Discovery

The discovery service is clearly a critical component of the proposed solution. In very simple scenarios, such as that described in section 3, it appears to be almost equivalent to a central controller, in that all of the participating peers perform most of their communication directly with this service. However:

- The discovery service is only required to match the initial subscriptions to the interaction roles. As the interactions become more complex, the proportion of interaction with the discovery service diminishes.
- The service is extremely lightweight and efficient, and can be easily replicated (the state is very simple).
- There are many different technologies which could be used to implement the service with varying characteristics - OpenKnowlegde and Okeilidh, for example, use completely different approaches.
- In a large, practical implementation, it is likely that the discovery service would be hierarchical. Local matches would be found quickly, and more remote matches would be forwarded to additional hubs. This appears to match the natural desire to solve negotiations locally and quickly when possible.

On a local scale, and with a comparatively low traffic rate, it it is possible to envisage a broadcast-based solution with no central service at all, but the discovery service approach is consistent with most agent-based systems which require some mechanism for participating peers to locate one another before the interactions can take place.

5.4 Federation

Apart from the issues of performance and scale, federation has been one of the main motivating factors for our approach – different organisations may have different services to offer, different requirements, and different restrictions on the information that they are willing to share and the peers with whom they are willing to interact.

A key feature of the OpenKnowledge approach is that there is no requirement for global agreement, either on protocols or ontologies – a group of peers can participate in an interaction simply by agreeing on the terms (constraints) used in that IM, and following the protocol that it provides. If a peer does not understand the terminology used by a particular IM, or is not willing to share the information that it requires, then it cannot participate in that particular interaction - but it is free to participate in other interactions, or even propose its own alternative model in which others can be invited to participate.

In particular, status or monitoring information is never shared or synchronised explicitly between the peers. A particular interaction may require knowledge of some specific parameter (say the network bandwidth available to the VM) in which case the interaction model will specify that participants must implement a constraint to determine this value, and be willing to share it.

Within one organisation, it is likely that the interaction models and OKCs will be curated centrally. In this case they will have been designed to interoperate, and the vocabulary used will be consistent. In a federated environment, IMs and corresponding constraints may be proposed by multiple organisations and it is necessary to understand how these relate, which are equivalent, and how we may map between them. For example, if two different interaction models both require us to provide the network bandwidth, do they use the same units? As we have already stressed, there is no requirement for a global agreement on such matters, but the OpenKnowledge ontology matching service proposes a solution to this problem by aggregating a number of techniques for ontological matching.

One other issue for large, federated systems is the potential performance degradation due to the larger number of potential participants, and the increased latency of interactions. This is an issue for both the discovery service, and the execution of the interactions themselves. In practice however, we would expect to see different kinds of interactions between local peers and remote ones – for example, we might expect a good deal of activity between local machines as they negotiate an efficient placement. But at some point, the local cluster may become overloaded and there may be an inter-site negotiation to transfer a block of machines into the cloud. This leads quite naturally to a hierarchical organisation of discovery services, and interaction models, suited to the locality of the communications.

The example in section 4 shows a basic workflow for this "cloud bursting" scenario. In practice, such an application is likely to require a more complex model: there may be significant dependencies between the virtual machines and the services running on them – for example, it may be necessary to make changes in firewall configurations. This clearly increases the complexity of the interaction models, although the basic principles remain unchanged. We are also presuming that it is possible to run agents on the physical machines within the data centre. This is clearly not the case for current commercial services such as EC2, for example. However, we could envisage running proxy servers representing such a service and managing the associated resources.

5.5 Configuration Patterns

In creating interaction models for various scenarios, it has become clear that there are some common interaction patterns. Perhaps the most obvious of these is the client-server pattern described in section 4 – there are many cases where a "service" of some sort needs to be moved, and this requires corresponding modifications to the client. Another pattern occurs when there is contention over some resource, and some further action is required to free up the necessary resource: for example, assume that we require a physical host for a big virtual machine which needs the full resources of one PM. If all of the available PMs are running small VMs, we may need to move one of the small VMs to create space for the large one.

Such patterns can be viewed in the same way as software design patterns and used to aid and clarify the manual construction of interaction models. But it may

also be possible to incorporate these into the tooling, by providing users with a higher level view of the system and allowing them to specify and compose such patterns explicitly. We are currently investigating the use of automated planning techniques [12,13] to compose such (parameterised) patterns automatically for managing complex workflows .

6 Related Work

There is a considerable amount of existing work on load balancing of virtual infrastructures. This usually involves a central service which collects monitoring data from the physical and virtual machines, computes any necessary reallocation, and orchestrates the appropriate migrations. Analysis of "hotspots"[14] or SLA violations[15] is necessary to plan a new allocation, but despite some success with statistical machine learning[16,17], effective prediction of *future* performance seems unrealistic in many cases. This type of centralised control limits the degree to which it is possible to exploit the resources of a more federated service[18,19]. Managing the interactions of imperative control algorithms in a centralised system is also a problem[20].

VMWare is a popular provider of commercial management infrastructure for virtual datacentres. The VMWare *vSphere Distributed Resource Scheduler* (DRS) product allows the user to specify rules and policies to prioritise how resources are allocated to virtual machines. DRS[13] "continuously monitors utilisation across resource pools and intelligently aligns resources with business needs" . *vSphere Distributed Power Management* (DPM) allows workloads to be consolidated onto fewer servers so that the rest can be powered-down to reduce power consumption. Citrix Essentials[14] and Virtual Iron "Live capacity" [15] are other commercial products offering similar functionality, and LBVM[16] is an open-source product based on Red Hat Cluster Suite. However, all of these products use a centralised management model.

Likewise, tools for managing the workflow of configuration changes are also standard practice, but based on a centralised execution model; IBM Tivoli Provisioning Manager[17] (TPM) is a common commercial solution with a workflow executed from a central control server. ControlTier[18], is a popular alternative which orchestrates the execution of the workflow by sending a secure shell remote command to the target node.

The term *autonomic computing*[21] was popularised by IBM in 2001 to describe computing systems which are *self-configuring*, *self-healing*, *self-optimising*,

[13] http://www.vmware.com/pdf/vmware_drs_wp.pdf
[14] https://h20392.www2.hp.com/portal/swdepot/displayProductInfo.do?productNumber=HPE4XSE
[15] http://www.storageengineers.com/pdf_virtualiron/Evaluation_Guide_0107.pdf
[16] http://lbvm.sourceforge.net/
[17] http://www.ibm.com/software/tivoli/products/prov-mgr/
[18] http://controltier.org/

and *self-protecting* (*self-**). Kephart and Walsh[22] noted that agent-based technologies are a natural fit for implementing this type of system, and this has led to the development of market-based resource management systems such as [23]. Several people have applied these techniques to virtual machine management: Xing[24] describes a system where "each virtual machine can make its own decision when and where to live migrate itself between the physical nodes" - for example, two VMs may notice that the applications running on them are communicating frequently, and the VMs may decide that they should attempt to migrate so that they are physically closer. Spata and Rinaudo[25] describe a FIPA-compliant system with very similar objectives to our own which is intended to load-balance VMs across a cluster. However, we are not aware of any other systems which are driven directly from a declarative specification of the interaction model.

7 Conclusions and Future Work

We have demonstrated that an agent-based approach using LCC interaction models is a viable technique for negotiating both virtual machine placement and execution of the associated workflows. This provides a framework for supporting arbitrary interaction models which are capable of implementing a wide range of policies and approaches, suitable for different situations. The interaction models clearly expose the protocols which can be easily verified, shared, composed and modified. A particular strength of this approach is the lack of any requirement for prior agreement on protocols or ontologies, which makes it a particularly effective solution in federated environments.

We are currently investigating more complex workflows, and particularly the automatic generation of interaction models using automated planning techniques.

Acknowledgments

This research has been partly supported by a grant from HP Labs Innovation Research Program Award.

References

1. Anderson, P., Bijani, S., Vichos, A.: Multi-agent negotiation of virtual machine migration using the lightweight coordination calculus. In: Proceedings of the 6th International KES Conference on Agents and Multi-agent Systems – Technologies and Applications. (2012)
2. Walton, C., Robertson, D.: Flexible multi-agent protocols. Technical report, University of Edinburgh (2002)
3. Robertson, D.: A lightweight coordination calculus for agent systems. Declarative agent languages and technologies II (2005) 109–115
4. Pinninck, A.P.D., Kotoulas, S., Siebes, R.: The OpenKnowledge kernel. In: Proceedings of the IX CESSE conference. (2007)

5. Siebes, R., Dupplaw, D., Kotoulas, S., de Pinninck Bas, A.P., van Harmelen, F., Robertson, D.: The OpenKnowledge System: an interaction-centered approach to knowledge sharing. In Meersman, R., Tari, Z., eds.: Lecture Notes in Computer Science. Volume 4803., Springer, Springer (2007) 381–390
6. Kotoulas, S., Siebes, R.: Adaptive routing in structured peer-to-peer overlays. In: 3rd Intl. IEEE workshop on Collaborative Service-oriented P2P Information Systems (COPS workshop at WETICE07), Paris, France, IEEE Computer Society Press, Los Alamitos. (2007)
7. Anadiotis, G., Kotoulas, S., Lausen, H., Siebes, R.: Massively scalable web service discovery. In: Advanced Information Networking and Applications, 2009. AINA'09. International Conference on, IEEE (2009) 394–402
8. Li, J.: Agent-based management of virtual machines for cloud infrastructure. Master's thesis, School of Informatics, University of Edinburgh (2011)
9. Trecarichi, G., Rizzi, V., Vaccari, L., Marchese, M., Besana, P.: Openknowledge at work: exploring centralized and decentralized information gathering in emergency contexts. (2009)
10. Bijani, S., Robertson, D.: A review of attacks and security approaches in open multi-agent systems. In: Artificial Intelligence Review. Springer (2012)
11. Osman, N., Robertson, D., Walton, C.: Dynamic model checking for multi-agent systems. Declarative Agent Languages and Technologies IV (2006) 43–60
12. Herry, H., Anderson, P., Wickler, G.: Automated planning for configuration changes. In: Proceedings of the 2011 LISA Conference, Usenix Association (2011)
13. Herry, H., Anderson, P.: Planning with global constraints for computing infrastructure reconfiguration. In: CP4PS-12 - The AAAI-12 Workshop on Problem Solving using Classical Planners. (2012)
14. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Black-box and gray-box strategies for virtual machine migration. In: Proceedings of the 4th Usenix Symposium on Networked Systems Design and Implementation, Usenix (April 2007)
15. Bobroff, N., Kochut, A., Beaty, K.: Dynamic placement of virtual machines for managing SLA violations. Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on (21 2007-Yearly 25 2007) 119–128
16. Bodk, P., Griffith, R., Sutton, C., Fox, A., Jordan, M., Patterson, D.: Statistical machine learning makes automatic control practical for internet datacenters. In: Proceedings of Workshop on Hot Topics in Cloud Computing (HotCloud). (2009)
17. Liu, X.: Prediction of resource requirements for cloud computing. Master's thesis, School of informatics, University of Edinburgh (2010)
18. Ruth, P., Rhee, J., Xu, D., Kennell, R., Goasguen, S.: Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. Autonomic Computing, 2006. ICAC '06. IEEE International Conference on (June 2006) 5–14
19. Grit, L., Irwin, D., Aydan, Chase, J.: Virtual machine hosting for networked clusters: Building the foundations for "autonomic" orchestration. Virtualization Technology in Distributed Computing, 2006. VTDC 2006. (Nov. 2006) 7–7
20. Schmid, M., Marinescu, D., Kroeger, R.: A Framework for Autonomic Performance Management of Virtual Machine-Based Services. In: Proceedings of the 15th Annual Workshop of the HP Software University Association. (June 2008)
21. Murch, R.: Autonomic Computing. 1 edn. IBM Press (2004)
22. Kephart, J., Walsh, W.: An artificial intelligence perspective on autonomic computing policies. In: Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on. (june 2004) 3 – 12

23. Schnizler, B., Neumann, D., Veit, D., Reinicke, M., Streitberger, W., Eymann, T., Freitag, F., Chao, I., Chacin, P.: Catnets deliverable 1.1: Theoretical and computational basis. Technical report, CatNet Project (2005)
24. Xing, L.: A self-management approach to service optimization and system integrity through multi-agent systems. Master's thesis, University of Oslo, Department of Informatics (May 2008)
25. Rinaudo, M.O.S..S.: Virtual machine migration through an intelligent mobile agents system for a cloud grid. In: Journal of Convergence Information Technology. Volume 6. Advanced Institute of Convergence Information Technology (June 2011)