Abstract—A promising approach to increase the efficiency of infrastructure usage is to adapt the assignment of resources to workloads. This can be used, for example, to consolidate existing workloads so that the new capability can be used to serve new requests, or alternatively unused resources may be turned off to reduce energy consumption. Many architectural solutions have been presented for data centre management, however these tend to be centralised and may suffer in their ability to scale and support data centres with tens of thousands of nodes. Distributed approaches solve the scalability problem, however these do not have a global view of resources across the data centre. To address this, we propose a novel hybrid distributed hierarchical framework that is effective at providing the information needed for decision making at scale. We evaluate the performance of our approach by simulation, and demonstrate that a hybrid approach is a viable solution for managing large data centres, through rapid information dissemination and ability to make decisions using a global view.

I. INTRODUCTION

Infrastructure Providers (IP) typically abstract data centre resources and present them to customers through a virtualisation layer, with a Virtual Machine (VM) as the most common form. A key challenge for IPs is to construct resource utilisation mechanisms that can scale to large data centres, which can house more than 100,000 nodes [1]. VM resource management for Infrastructure as a Service (IaaS) entails controlling the admission of new VMs, onto physical nodes and periodically adapting resource configuration to achieve a management objective. Proposals in the literature aim to minimise Service Level Agreement (SLA) violations, and some trade this off with another objective, such as reducing energy consumption or maximising IP revenue [2]. To achieve these objectives, several approaches are used, which typically rely on a central management controller for deciding how to change the configuration of the infrastructure. However, the architecture of the management controller has a significant impact on the scalability of a proposal, with centralised approaches not being able to scale to large data centres [2], as CPU and memory adaptation are scheduled across the entire infrastructure. While this gives opportunities for global optimisation, it presents a significant challenge when managing thousands of resources. The central controller in [3] took significant time to execute the scheduling of 15 nodes, which had $10^8$ control options, just to adapt CPU resource; memory configuration was not covered. The centralised engine in [4] was only able to manage 400 nodes with 1000 VMs, when adapting CPU and VM configurations.

In order to solve the scalability challenge of large data centres, hierarchical and distributed approaches have emerged as an alternative and received considerable attention recently [5], [6], [7], [8], [9], [10].

Hierarchical architectures tackle the scalability challenges of centralised approaches, by dividing the infrastructure into several levels and lowering the number of nodes handled by the management controller, thus reducing the time complexity of solving a global optimisation problem. The challenge with hierarchical architectures is dividing the infrastructure into small enough groups to enable efficient local resource allocation while remaining coordinated enough to come close to a globally optimal solution. Distributed architectures are decentralised and typically do not use a central decision making component. They enable individual nodes to cooperate directly [7], [8], [9], [10], and thus allow the approach to scale to very large data centres. One of the challenges with distributed architectures is they typically consume a high amount of bandwidth for the communication between the nodes across the whole data centre. In addition, due to reduced visibility of the whole infrastructure, distributed approaches tend to make locally plausible but globally suboptimal decisions.

We propose a Scalable Hierarchical Distributed Framework (SHDF), which overcomes the weaknesses of both approaches, by achieving high scalability and ability to apply efficient allocation. SHDF consists of hierarchical controllers operating at different scopes. On the lowest level, every node in the infrastructure contains a Node Controller (NC), which dynamically adjusts resource configurations to satisfy VM demands on each node. Each NC operates in a distributed architecture and cooperates with a Lead Node (LN), which is a higher level controller for all the NCs within a given cluster. Our contributions are:

1) An architectural framework for data centre infrastructure, which can manage a large cloud data centre spanning thousands of nodes.

2) A new hybrid architecture, which has the scalability characteristics of distributed systems and the ability to construct a global view.

3) An empirical evaluation of the hybrid architecture, in comparison with hierarchical and distributed approaches.
The rest of this paper is organised as follows. Section 2 describes background and related work. Section 3 describes the architecture of our proposed scalable framework and Section 4 describes an implementation of the architecture. Section 5 presents an evaluation of our implementation and compares it to two other approaches: distributed and hierarchical. In Section 6 we discuss future work and draw conclusions.

II. BACKGROUND & RELATED WORK

1) Hierarchical Controllers: Hierarchical approaches have been widely studied in cloud resource management. The approaches typically use a multi level hierarchical approach running at different time intervals. In Addis et al. [5] the lowest level controller runs every hour and performs VM placement, power management and workload profiling. The authors claim this can scale to 7,200 nodes with up to 60,000 VMs. In [11], the lowest level controller manages a small number of machines and the applications that are hosted on them. At the next higher level, a controller manages machines owned by multiple lower level controllers. The authors in [12] used 3 levels, where the highest level controller managed multiple clusters operating at seconds (L1), minutes (L2) and days (L3) intervals, however the authors did not explore the scalability of their approach. The authors in [13] chose to slice the hierarchy along the operations of the controllers. A Level 1 controller handles VM placement and load balancing, and runs every 30 minutes. A Level 2 controller handles the resources of a node, and runs every few minutes. The authors in [14] focus on VM placement in their approach using a 2 level hierarchical controllers.

Hierarchical proposals typically utilise a controller running in a centralised manner within the scope of a cluster of nodes. As the number of nodes within the cluster increases, the decision making algorithm faces the same challenges as traditional centralised approaches, with the execution time resulting in an inability to react to SLA violations. Our proposed framework tackles this by reducing the operations performed by the centralised controllers and distributing management functionality to lower execution nodes, therefore providing a hybrid hierarchical distributed framework. In our approach, each node controller autonomously manages the execution node, including detection of stress states and violation of SLAs. These node controllers can cooperate with other nodes and higher level controllers to perform management of the infrastructure and workloads.

In [15], the authors use a hierarchical approach with VM migration escalation. The approach performs initial assignment of VMs to clusters (containers) and periodically, lower controllers decide what to optimise and pass the decision to parent controllers. The authors consider the time complexity of the decision making algorithm and place an upper bound of optimisations to be performed by the hierarchical controllers.

In [16], the authors outline how a collection of hierarchical autonomic managers can collaborate using messages. While the authors acknowledged the importance of scalability, they did not explore the scalability of their approach.

Our approach has similarities to [17], which proposed a framework for managing a hierarchical cloud management system. Due to the computational requirements of management nodes, the authors propose every node is either used as a management node, or an execution node, never playing both roles. This contrasts with our approach, execution nodes can also perform management roles, resulting in an fewer nodes dedicated to the management of the infrastructure. Our approach additionally enables the leaf nodes to operate in a distributed manner. Proposals typically do not build cooperation between controllers, resulting in controllers operating independently. In contrast, the authors in [12], [18], [19] proposed controller cooperation. In [12], [19], the lower level controllers propagated workload satisfaction with assigned resources to a higher controllers, which is then used in to possibly further optimise resource assignment. The authors in [18] proposed cooperating controllers, where lower level controllers can escalate a management request to a higher controller, rather than waiting for an action in the next management cycle. However the authors approach only escalated from a mid level controller within the hierarchy and did not escalate from the lowest level controllers, which our approach does.

2) Distributed Controllers: In [20], [8], the authors use a heuristic implemented as a peer-to-peer protocol, enabling nodes to communicate directly without a centralised controller. Two cooperating nodes determine whether to migrate a VM based on the defined objectives. While [20] did not take into account the cost or duration of the conflict before applying the migration, [8] incorporated migration cost into the decision making. A periodic node discovery service enables nodes to find new neighbouring nodes to communicate with. On each round of the protocol, two cooperating nodes determine to migrate a VM based on defined objectives. The distributed approaches are used to redistribute the load across the cluster as well consolidate VMs. Using simulation, the authors claim their approaches can manage more than 100,000 nodes. A challenge with distributed approaches is the lack of a global view of the infrastructure, which impacts the ability to reach a globally optimal solution. In contrast, our approach has the scalability characteristic of distributed systems yet has the ability to have a global view of the infrastructure through controllers at each level of the hierarchy.

The authors in [10], proposed a distributed probabilistic algorithm, using an overlay network and a decentralised load balancing technique. The authors considered scalability and impact of node availability churn, although they did not discuss the impact on energy consumption. The authors proposal is a distributed algorithm and can be integrated with our proposed SHDF.

The authors in [21] proposed a distributed approach for managing the workload of large, enterprise cloud data, focusing on reducing energy consumption and SLA violations. The authors used a hypercube structured overlay, with similar cost to our approach, with N nodes reaching status propagation in O(log N) rounds. The authors in [27], proposed a distributed
scheme where nodes broadcast resource requests to all nodes, during a migration scenario. In contrast, our approach has a bigger view of the infrastructure, through the additional hierarchical controllers, and can consolidate nodes across a whole cluster.

The authors in [9], proposed a distributed self-organising approach, where nodes cooperate within the overlay. Migration decisions are performed after an evaluation of the whole overlay state. Similar to our approach, nodes collaborate across the overlay, however the close physical proximity of cooperating nodes in our approach means gossip traffic and migration traffic is localised within the cluster. Similar to us, the authors also concluded that a centralised view is able to achieve better consolidation results, as it has a global view of the infrastructure.

To the best of our knowledge, SHDF is the first hybrid hierarchical distributed framework that enables leaf nodes to operate in a distributed manner, and combines this with hierarchical allocation and consolidation of VMs across large sections of the infrastructure.

III. ARCHITECTURE

A. Framework components

The management of nodes in an IaaS environment can be abstracted as 2 dimensions, Management Algorithm (MA) and Management Framework (MF). The MA is responsible for deciding how workloads are assigned to infrastructure resources, while the MF enables the MA to execute by providing common functionality, such as hierarchy level management and aggregation of metrics between nodes. The combined functionality results in workloads executing on infrastructure nodes. In the following sections we briefly describe the MA and detail our proposal, which is in the form of a MF.

B. Management Algorithm (MA)

In our previous work [2], we have shown management algorithms (MAs) are widely covered in the literature, and drive the decision making process in cloud systems adaptation. Several analytical techniques are used, including Control Theory, Heuristics and Machine learning. Cloud systems adaptation needs to be invoked in order to evaluate the infrastructure and determine whether resource reconfiguration is required. The approaches used in the literature fall onto reactive and proactive engagement. Reactive approaches invoke adaptation at defined time intervals or when a monitored metric, e.g. CPU utilisation, reaches a specific threshold. Proactive approaches predict what demands will be placed on the infrastructure and invoke adaptation ahead of the predicted resource contention point. The MA assigns resources in the infrastructure and regularly assesses the satisfaction of such assignments in achieving a given Service Level Agreement (SLA). The frequency of this assessment is influenced by the time complexity of the algorithm; the lower the complexity, the more frequently the algorithm can be executed. Our proposed architecture enables the MA to run more frequently, and is described in the next section.

C. Management Framework (MF)

The MF provides common utilities that enable the MA to execute, including a mechanism to propagate node state, and a decision engine architecture that may be centralised, hierarchical or distributed. The MF is the focus of our work in this paper.

The architecture of our proposed framework SHDF, as shown in Figure 1, consists of hierarchical controllers operating at different scopes. On the lowest level, every node in the infrastructure contains a Node Controller (NC), which dynamically adjusts resource configurations to satisfy VM demands on each node. A collection of NCs form a cluster of nodes. Each NC cooperates with a Lead Node (LN), which is a higher level controller for all the NCs within a given cluster. Unique to our proposal, the NCs within each cluster are divided into logical groups, called overlays, where a NC cooperates with other NCs within the same overlay. Each NC exists in only one overlay and in one cluster. This architecture enables nodes to cooperate in a hybrid hierarchical distributed manner.

Unique to our approach, the LN operates as a normal node within the infrastructure in addition to its management responsibility towards the cluster. A collection of clusters creates a level, $n$, which is managed by a controller at level $n+1$. At the highest level, the Data Centre Controller (DC) manages the controllers one level below it.

SHDF attempts to service resource requests at the lowest local level possible, in order to reduce the overhead of servicing the request [22] and to reduce the performance impact of migrating VMs across cluster boundaries [6]. All the controllers in SHDF manage multiple execution nodes, with the exception of the Node Controller (NC), which manages a single node.

The following subsections describe the components of SHDF and the functionality it provides to the MA.

---

Fig. 1: Hierarchical Escalation Architecture
TABLE I: Exchanged data between overlay nodes

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NodeID</td>
<td>The ID of the node, data belongs to</td>
</tr>
<tr>
<td>VMSize[NumberAvailable]</td>
<td>number of predefined VM sizes that can be hosted on this node</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Time stamp for captured metrics</td>
</tr>
</tbody>
</table>

1) Overlay Management: Execution nodes operate in a distributed way within the boundary of a cluster and are organised in multiple overlays. An overlay is a fundamental concept to reliable multicast, as it abstracts the details of the underlying physical network by building a virtual network on top of it, which can be seen as a graph that represents a link between nodes. To construct overlays, typically two main approaches exist [23]: structured and unstructured protocols. Structured approaches tend to be efficient in terms of number of links, but are sensitive to node failure, because the overlay structure takes into account metrics such as latency or bandwidth, and upon node failure the structure needs to be rebuilt. In unstructured approaches, links are established randomly among the nodes. To ensure overlay stability, multiple links are established to ensure redundancy, however this can cause nodes to receive multiple copies of a given message through its different neighbours.

SHDF is closer to structured approaches [24], where the overlay is constructed to span a subset of a cluster, and by definition is layered on a physical structure of nodes, which solves the proximity challenge in structured approaches, without needing to associate latency metrics with each of the overlay links. When a new node is added to the infrastructure, it gets added to a cluster and assigned an overlay. SHDF enables the MA to determine where a new node is added. When a node is added to an overlay, it receives details of neighbouring nodes from the same overlay, through the data dissemination mechanism described below.

2) Data dissemination: In centralised and hierarchical architectures, a central component receives state updates from all the nodes under its management, to enable the MA to adapt the infrastructure. SHDF removes this dependency and enables the nodes to exchange their state within an overlay. Each node holds a local cache of the state of other nodes in the overlay, and to populate this cache, nodes exchange state messages. A node selects another node at random from the overlay and exchanges its view of the overlay. We use a pull & push gossip approach, where node x sends new state updates to node y, and retrieves new states updates at node y. A new node state takes O(log N) rounds to reach all nodes, where N is the number of nodes in the overlay [25]. A gossip round completes when every node has initiated an exchange exactly once. The populated local cache allows the MA to perform VM migration to other nodes within the overlay, and allows the LN to perform VM consolidation across the cluster.

Similar to [7], a NC regularly exchanges state with other NCs and holds a cache of free capacity of other cooperating controllers in the same overlay, shown in Table I. The gossiping algorithm is shown in Algorithm 1. We enhance this further and make nodes gossip, when there is a state change such as inward migration or a node is being shutdown. Each node will keep the latest metrics seen, and every time an exchange of state occurs the timestamp is used to decide if each exchanged entry is later than any existing entries. If it is, the node’s internal state is updated, otherwise the data exchanged is discarded. The MA running on each node can use these metrics to enable nodes to cooperate in resource allocation requests, which will be described later.

Algorithm 1 stateExchange@Node

1: procedure EXCHANGESTATE(localState)
2:   overlay ← getOverlayMembers - thisNode
3:   randomMember ← random(overlay)
4:   remoteState ← randomMember.gossip(localState)
5:   for i in remoteState do
6:     remote ← remoteState[i].timeStamp
7:     local ← localState[i].timeStamp
8:     if remote > local then
9:       localState[i] ← remoteState[i]
10:  procedure GOSSIP(remoteState)
11:     respond to caller ← getLocalState
12:     receivedState ← remoteState
13:     for i in receivedState do
14:       received ← receivedState[i].timeStamp
15:       local ← localState[i].timeStamp
16:       if received > local then
17:         localState[i] ← receivedState[i]

As in Figure 2, the Lead Node is an ordinary node, it will also exchange its own state. The LN is a member of all the overlays in the cluster and will receive state exchanges from all the nodes within the cluster. To restrict the view of a node and reduce redundancy of state management, when a node gossips with the LN, it will only receive state updates from its overlay and not from other overlays the LN knows about.
SHDF enables the LN to send aggregated metrics for all the nodes within its cluster, to other cooperating controllers. The LN participates in an overlay with other LNs, in a similar way to the NCs, and it can exchange aggregated metric data. The parent controller to the LN is always included in each LN overlay, resulting in state exchanges between the LN in an overlay and the managing controller, in a similar way to the exchanges between a LN and a NC, as shown on Figure 3. The aggregated data can be used by controllers at higher levels in the hierarchy, resulting in an aggregated view of the whole infrastructure to be available at the DC controller level.

3) Controller functionality - VM migration: When a node cannot satisfy demands of the VMs it hosts, it starts an escalation process that aims to resolve the request at the lowest possible level. The MA running on the stressed node and the LN need to cooperate to resolve the escalated VM migration, by using our provided framework mechanisms.

Table II shows the available methods to support migration and escalation of a VM. SHDF uses different methods to migrate within the overlay and escalate outside it, with a higher priority for escalation. The NC can send a migration request to other nodes within the same overlay, by using the accumulated metrics from other nodes. In turn, the MA running on the LN can query the cluster records from all the overlays, which has state data from all nodes in the cluster, to find a suitable node to house an escalated VM. When there is no free capacity in a single node to match the VM request, but there are enough free resources across the whole cluster, the LN can choose to invoke a consolidation process, where multiple VMs could be migrated between nodes to facilitate creation of enough space to house an escalated VM.

If it is not possible to house the VM escalation within the cluster, the MA running on the LN can use the aggregated metrics, of free capacity, from other LNs and decide to forward the escalation to another LN in the same overlay as the originating LN. If no suitable node is found, the MA can escalate the request to a higher level controller (LN parent) with a broader view of the infrastructure. This is repeated until a node is found or the escalation reaches the DC Controller, which has a view of the entire data centre, and the MA on the DC can choose to direct the request to other parts of the hierarchy, by utilising the aggregated data received by the DC controller. As requests progress through the escalation process, they are assigned an increasing priority, which can be used by the MA in the decision making process. For example the MA may choose to prioritise finding a host for an escalated VM compared to a new VM placement. If the aggregated metrics at the DC show no availability to house the escalated VM, then the request is rejected and the originating node is notified.

TABLE II: Controller functionality for Migration of VMs

<table>
<thead>
<tr>
<th>Method</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migrate VM</td>
<td>Node a</td>
<td>Node b</td>
</tr>
<tr>
<td>Escalate VM</td>
<td>Node a</td>
<td>LN</td>
</tr>
<tr>
<td>Place Escalated VM</td>
<td>LN</td>
<td>Node b</td>
</tr>
<tr>
<td>Escalate VM</td>
<td>LN a</td>
<td>Level n</td>
</tr>
<tr>
<td>Escalate VM</td>
<td>Level n Controller</td>
<td>Level n +1 Controller</td>
</tr>
<tr>
<td>Escalate VM</td>
<td>Level n Controller</td>
<td>DC</td>
</tr>
<tr>
<td>Place Escalate VM</td>
<td>DC</td>
<td>Level n -1 Controller</td>
</tr>
<tr>
<td>Place Escalate VM</td>
<td>Level n Controller</td>
<td>Level n -1 Controller</td>
</tr>
<tr>
<td>Place Escalate VM</td>
<td>Level n Controller</td>
<td>LN</td>
</tr>
</tbody>
</table>

4) Controller functionality - Consolidation: At periodic time intervals and changes in utilisation, each of the management controllers, and LNs, can invoke a consolidation process where the MA can examine the state of the infrastructure and for every node under its management, decide to:

- Migrate some VMs from a node
- Migrate all VMs off a node and switch the node off
- No change

The advantage of SHDF is it allows the nodes to primarily operate in a distributed manner for time sensitive operations such as VM migrations. For non time sensitive operations like VM placement and consolidation, SHDF enables the running of time complex algorithms across parts of the infrastructure or even the whole infrastructure. Compared to distributed approaches, this will enable the MA to have a much more comprehensive view of the infrastructure during these operations.

5) Boot strapping & Hierarchy Management: We defer development of the some of the functionality of SHDF, including ability to bootstrap, dynamically adapt the hierarchy, high availability of the LN and clock synchronisation to further work in the future.

IV. IMPLEMENTATION OF SHDF

In order to evaluate SHDF, we combined it with a Management Algorithm (MA) to utilise the capabilities of our
proposed architecture. The following is a description of the MA components. To focus our evaluation on the capabilities of the MF and not the features of the MA, we choose a simple MA from other work in the literature [18], which has been cited by other researchers, and uses a threshold-based approach to engage adaptation of the infrastructure.

SHDF design enables several levels of hierarchy, with controlling managers at each level. To simplify initial experiments, we choose the smallest number of levels, and instantiated SHDF with 3 levels of management with NCs at Level 1, LNs at Level 2 and a Data Centre controller at Level 3. We intend to implement and evaluate more levels in the future.

A. VM Migration

As in Algorithm 2, we use the utilisation metrics available to the NC and send a migration request to candidates in the same overlay, marking them as tried, and wait for a response. The NC sorts candidate targets in the following order: Partially utilised (<upper threshold), used utilised (<low threshold) and empty (0 VMs), and tries them in this order, to achieve higher node utilisation and reduce selection of powered-off nodes.

Algorithm 2 Create VM Migration@Node

1: \( x \leftarrow \text{chooseVMConsumingMostResource()} \)
2: \( \text{sort}((\text{PartialUtilisation}, \text{lowUtilisation}, \text{empty}) \)
3: \text{for} \; k \; \text{in} \; \text{nodeNeighborList} \; \text{do} \\
4: \quad \text{if} \; k \; \text{canHost} \; x \; \text{AND notTried} \; k \; \text{then} \\
5: \quad \text{send migrationRequest to} \; k \\
6: \quad \text{mark} \; k \; \text{tried} \\
7: \quad \text{return} \\
8: \text{EscalateToLeadNode()} \\

Once a candidate is marked as tried, we do not use it again until we see an updated state from it through the gossiping mechanism. The NC tries candidates until it gets a response or we exhaust candidates in our overlay.

When the LN receives the escalated request, as in Algorithm 3, it will be able to forward the request to all eligible nodes within the cluster. As this is an escalated request and it came from within an overlay within the cluster, the sending overlay is excluded from the list of eligible target nodes.

Algorithm 3 Process VM Migration@LN

1: \( \text{req} \leftarrow \text{migrationRequest} \)
2: \( \text{nodes} \leftarrow \text{clusterNodes} - \text{sendingOverlayNodes} \)
3: \( \text{sort}(\text{nodes}, \text{PartialUtilisation}, \text{lowUtilisation}, \text{empty}) \)
4: \text{for} \; k \; \text{in nodes} \; \text{do} \\
5: \quad \text{if} \; k \; \text{canHost} \; \text{req} \; \text{AND notTried} \; k \; \text{then} \\
6: \quad \text{send req to} \; k \\
7: \quad \text{mark} \; k \; \text{tried} \\
8: \quad \text{return} \\
9: \text{EscalateToDC} \\

When a node gets an escalated migration request, via the LN, it processes it in a similar way to requests from within the overlay. If it can be accommodated then an accept request is sent to the original sender, otherwise a reject request is sent to the LN. The LN will keep forwarding the migration request to nodes within its cluster until it has tried all nodes, and if it fails to place the migration, it will escalate the request to the DC controller, one level above it. The DC Controller will search for a cluster that can host the VM based on aggregated metrics from all the LNs.

A DC Controller selects the first target cluster that can host the escalation, and forwards the request to the LN responsible for that cluster. When the identified LN receives the escalated migration request, it processes it in a similar way to escalation from within its own cluster, in that it identifies candidate nodes and forwards them the request. If the receiving LN cannot host the migration within its cluster it replies with a rejection to the DC, which in turn will attempt other clusters until it exhausts all clusters. At this point we have exhausted the entire infrastructure and failed to find a suitable node to house the VM migration. The DC will then send a final reject message to the original migrating node.

B. VM Placement & Consolidation

We implement the VM placement and consolidation approach in [18], and utilise the SHDF architecture to disseminate metrics to the LN and DC.

V. EXPERIMENT SETUP AND EVALUATION

We use simulation to facilitate rapid development of experiments of large data centres. We selected DCSim [26] because of its extensibility and existing implementations for hierarchical and distributed approaches, allowing us to create baseline comparisons for our proposed MF. To focus our evaluation on the features of the architecture and not the features of the MA, we implemented the MA from [18] with SHDF. We also utilised the existing proposals from [18] for the hierarchical and [27] for the distributed baseline comparisons. Source code for both approaches is readily available and has been cited by other researchers. The baseline MAs are similar, and are based on a greedy heuristic, that reacts to utilisation thresholds. While the MA used in the comparison does not include several features, such as workload prediction and migration cost, using the same MA allows us to assess the capability of the MFs.

We instantiate both SHDF and the hierarchical approach [18] with 3 levels of controllers, running on the root of the data centre, the cluster manager and leaf nodes.

A. Simulator Setup

DCSim allows a VM to use more CPU than reserved, up to an amount that does not impact other VMs. Like [18], we use a CPU utilisation thresholds of 90% for high, indicating stress level, and we use 60% for low, indicating low utilisation.

In DCSim, an application is modelled as an interactive multi-tiered web application. Each application has a specified client think time, and a workload component. The workload
defines the current number of clients connected to the application, which can change at discrete points in the simulation based on a trace file. The resource requirements are defined as its resource size, which is the expected amount of: CPU, memory, bandwidth and storage. DCSim treats bandwidth and storage as fixed requirements, however, CPU requirements can be varied across the simulation based on the VM demands. The DCSim energy model defines how much energy the node consumes at different CPU utilisation levels, and is calculated using results from the SPECPower benchmark. The benchmark provides energy consumption levels of real servers in 10% CPU utilisation intervals, and DCSim uses these values and calculates intermediary values using linear interpolation [26]. We assume that CPU energy consumption is the most contributing factor to a node’s energy consumption [28] and we use the default DCSim energy model in our experiments. DCSim applies a cost to VM migration including the time taken for migration, as a function of memory consumed by a VM, and factors the bandwidth required for the VM migration on the hosting node. Additionally, the boot time of a switched off node has an elapsed time cost. The time taken to switch on a node and migration is reflected on the time period the VM is in a stressed state, and therefore the SLA achieved by a VM.

For the gossip protocol, the exchange frequency and fan out values influence propagation speed and reliability. The fan out value determines the number of times a new state is exchanged by a node. For example, when a node receives an update and the fan out value is 2, the node will exchange this newly received state in the next 2 cycles of the gossip protocol. We trade bandwidth consumption with propagation speed, and chose a gossip frequency of 2 minutes and fan out value of 2. Due to limited space, we omitted our analysis of these parameters.

1) Workload & SLA: We run the experiments at a load that requires on average 80% of the CPU resources of the data centre. Each simulated application contains a workload trace based on the number of incoming requests to web servers from publicly available traces. Due to limited space, we limit our investigation to the google_corest_job_type included with DCSim and create VMs that demand 4 cores, 4.5GHz core capacity and 2GB RAM. DCSim is able to track SLA violations based on the response time of VMs, and we set this to trigger any time the response time is above 1 second.

2) Data centre: Our experiments use nodes modelled as ProLiant DL160G5 [29], with 2 quad-core 2.5GHz CPUs and 32GB of memory. The number of nodes used is specified in each of the experiments, and is typically up to 50,000 nodes. We assume that the data centre supports live VM migration, as this technique is currently supported by most major hypervisor technologies, such VMware [30] or Xen [31].

The various parameters used in our evaluation are outlined in Table III.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low threshold value (triggers consolidation)</td>
<td>60% CPU utilisation [18]</td>
</tr>
<tr>
<td>High threshold value (triggers migration)</td>
<td>90% CPU utilisation [18]</td>
</tr>
<tr>
<td>Node Monitoring Period</td>
<td>2 minutes [18]</td>
</tr>
<tr>
<td>Consolidation Frequency</td>
<td>Every 1 hour</td>
</tr>
<tr>
<td>Size of Overlay</td>
<td>100 nodes</td>
</tr>
<tr>
<td>Gossip Frequency</td>
<td>2 minutes</td>
</tr>
<tr>
<td>Fan out value</td>
<td>2</td>
</tr>
<tr>
<td>Delay before node is switched on, from Off State (impacts SLA)</td>
<td>3min</td>
</tr>
<tr>
<td>SLA Threshold</td>
<td>1 Second</td>
</tr>
<tr>
<td>Simulation Duration</td>
<td>3 Days</td>
</tr>
</tbody>
</table>

B. Evaluation: SHDF settings

The frequency at which the gossip protocol executes influences the speed of data dissemination. Figure 4 shows the impact of running the gossip protocol at different time intervals on 5 clusters of 1000 cooperating nodes, 5000 total. The more frequently the protocol runs, the higher the traffic consumption, as shown by figure 4a, which is the trade off between delivering fresh metrics and traffic consumption. Any MA combined with SHDF can configure the frequency of this setting. Encouragingly the scalability of central components remained consistent (milliseconds variation) as the protocol executed more frequently, as shown in Figure 4b. This shows SHDF has a low overhead as MF.

We have also run experiments with different parameters for size of Overlay and fan out values. Larger overlay sizes consume larger amount of MF traffic, and the scalability remained constant. Increasing the fan out value also increased the amount of consumed management traffic. We have omitted these results due to limited space.

C. Evaluation: Scalability

Scalability bottlenecks are typically found in centralised access to resources, with distributed systems improving scalability by parallelising computation and resource access. To evaluate the scalability of both the hierarchical approach and SHDF, we measure the time cost of central components on both approaches. Distributed approaches typically do not have a central migration component and thus are not relevant in this comparison. For the hierarchical approach, the central migration components are: cluster metric aggregation, node stress detection and VM escalation. For SHDF, the central migration components are: cluster metric aggregation and VM escalation. As SHDF is using the same MA as the hierarchical approach, examining the time in central components will allow us to compare the features of the MF. Figure 5a shows SHDF spends milliseconds in central components, compared to 100s of seconds by the hierarchical MF, resulting in an inherently scalable architecture. The hierarchical approach on the other hand, had a near linear increase in central components, suggesting a high fixed cost as the number of nodes increase, which is due to the central processing of metrics and actioning migration on behalf of every stressed
node. For the largest experiment of 50,000 nodes, SHDF was configured with a NC in each node and 50 LNs, each managing a cluster of 1000 nodes. Both NCs and LNs are execution nodes and only the DC Controller is modelled on a dedicated management node. In contrast to the framework in [17], where nodes are either execution or management nodes.

The increase in time cost between 30,000 nodes and 40,000 in 5a is due to how the MA [18], used by SHDF, uses a linear search to find available nodes during an escalation. This can be optimised by periodically sorting the available nodes, which will reduce the search time.

The MA [18] implemented by both the hierarchical approach and SHDF uses a greedy heuristic to find a migration target, and many alternatives in the literature are more complex. The authors in [32] incorporated the cost of migration and [11] incorporated power cost of decision engine in the migration. Additionally, proactive approaches [33], [11] start the VM migration before the conflict occurs, and in [11] the authors proposed performing a cost and benefit analysis before applying migration, and only invoked a migration if the benefit outweighed the cost of the migration. All of these methods to improve the quality of the decision making increase the sophistication of the MA, and combining these techniques with our proposed MF, should result in a highly scalable architecture, and will allow the MA to be deployed in a scalable framework, even when used to manage a large data centre.

D. Evaluation: Management Framework Traffic

In all of the evaluated approaches, each MF propagates node status to allow the allocation of resources. In the Distributed approach, node status is sent when one of the nodes is stressed and is looking for a migration target and when nodes are cooperating to perform consolidation. This status propagation occurs across the whole data centre, and as shown in Figure 6d, the distributed approach consumes the most management traffic out of all three approaches. In the hierarchical approach, leaf nodes send their status to one higher level controller and do not share their status with other nodes, which results in the least management traffic, at the cost of total reliance on a central component for decision making. In SHDF, nodes exchange their status within the same overlay, to enable rapid cooperation and decision making between the nodes. A node gossips its state at regular intervals and when a VM is migrated into or off a node. Thus during high load scenarios, as in this experiment, SHDF will consume more management traffic in order to propagate status state within the overlay.
E. Evaluation: SLA & Energy Usage

Critical to the effectiveness of a MF is to enable the MA to achieve SLAs and balance this with energy consumption of switched on nodes. Using end to end metrics like SLA and energy consumption, allows us to evaluate the influence of the MF, when combined with the same MA.

Figure 6a shows the total number of created VMs, the ideal case at 100% achievement, and the achieved SLA level for the three approaches. Using a similar MA to the hierarchical and distributed approach, SHDF meets more SLA targets than both the hierarchical and distributed approaches, as nodes perform their own stress detection and have a local view of the overlay, which enables each node to make rapid decisions about migrating VMs to an available target node. In the distributed approach, each node also does its own stress detection, but nodes do not have a local cache of the state of other nodes in the data centre. When stress is detected, a node broadcasts a request to migrate a VM away and waits for and then processes responses, resulting in VMs waiting for longer before they are migrated away from stressed nodes, with most VMs not achieving a 99% SLA. Even at 95% SLA achievement level, the distributed approach remains impacted by the lack of rapid decision making, compared to SHDF and hierarchical approaches, and many VMs do not achieve their SLA.

The number of migrations carried out by all of the approaches, Figure 6b, highly correlates to the VM SLA achievement, Figure 6a. Both the hierarchical approach and SHDF migrated VMs several times in order to reduce SLA violations, with approximately 6 migrations per placed VM. The reason for this, is the MA used by both the hierarchical approach and SHDF is a simple heuristic that does not predict workload patterns and engages adaptation reactively, and can cause system oscillation through large number of migrations. In the distributed implementation, once a migration request is broadcasted, the requesting node has to wait for and process the responses. This results in an elapsed time, which reduces the number of executed migrations and the achieved SLA. The hierarchical approach consolidates VMs at the rack level. It will search for under utilised nodes, move their VMs to other nodes in the rack, and switch the under utilised nodes off. SHDF performs a similar role across the cluster level, and as a cluster typically contains several racks, SHDF has a higher number of target nodes. We see the impact of this in Figure 6c, where both approaches consume a similar amount of energy, with SHDF being able to keep nodes switched off for longer. The distributed approach is not able to achieve VM demands quickly enough and thus incurred SLA violations, and in turn it did not consume as much energy as the other approaches.

VI. Future Work and Conclusion

We presented SHDF for managing cloud data centres, in which the nodes of the data centre are physically organised in the typical cluster formation and leaf nodes are then able to cooperate within logical boundaries of an overlay.
Additionally, the framework allows this pattern to be repeated at higher levels in the hierarchy. SHDF enables a MA to solve the allocation of resources and to configure the attributes of SHDF. Evaluation of SHDF shows its ability to retain the properties of a chosen MA, while increasing scalability by reducing time spent in centralised components. Further areas of research include adding adaptability to the framework to allow it to adjust and fine tune its parameters dynamically. For example, in a high load scenario, it might be more optimal to increase the propagation of the gossip exchange to allow nodes to have a fresher view of nodes in the overlay. In contrast, in low load scenarios, nodes can conserve bandwidth by lowering the frequency of gossip exchanges.

Further steps are necessary to investigate the impact of node failure on our gossip protocol, and its ability to continue propagating state changes within the overlay.

REFERENCES


