A Scalable Approach for Structuring Large-Scale Hierarchical Cloud Management Systems

Hendrik Moens and Filip De Turck
Ghent University – iMinds, Department of Information Technology
Gaston Crommenlaan 8/201, B-9050 Gent, Belgium
e-mail: hendrik.moens@intec.ugent.be

***Abstract*—In recent years, the scale of clouds and networks has increased greatly. It is important to ensure that the management systems used in these environments can scale as well. A centralized system does not scale well, while for distributed approaches, it is difficult to maintain an overview of the global system state. In hierarchical management systems, nodes at a low level in the hierarchy have a detailed view of a small part of the network, while higher-level nodes have a less detailed view of larger parts of the network. This makes hierarchical management systems well suited for large scale systems. The structure of such a hierarchical system should however be impacted by the management system for which it is used, as various properties such as the number of child nodes, tree depth and the distance between nodes can impact the performance of the management system. In this paper, we describe the Scalable Hierarchical Management Framework (SHMF), a scalable approach for constructing a hierarchical management system, suitable for large-scale cloud environments, that automatically optimizes its structure in function of its overlying management system. We evaluate the approach based on the requirements for the cloud application placement problem.**

I. Introduction

When building management systems for large scale environments such as networks and clouds, it is important to design them ensuring they scale well. Despite this, it is desirable for the global system state to be known, ensuring it can be monitored, and its operation can be optimized. These goals conflict, and are difficult to combine. Centralized management systems do have a good system overview, but they scale badly as the number of nodes increases. Conversely, distributed approaches do scale well, but it becomes more difficult to determine and optimize the global system state.

By making use of hierarchical management systems, properties of both approaches can be combined, making them useful for cloud management: different levels within the hierarchy have a different view of the entire management system state. Nodes low in the hierarchy have a detailed view of a small part of the system, while nodes higher up in the hierarchy have a less detailed view, but they view a much larger part of the system. These management systems scale well [1], as the load of managing the infrastructure is divided over multiple nodes that are structured hierarchically. At the same time, the nodes higher up in the hierarchy still achieve an overview of the entire system. We refer to inner nodes in the management tree as *management nodes*, that are used to manage the application, while *execution nodes* are the leaf nodes of the tree that are managed by the management system.

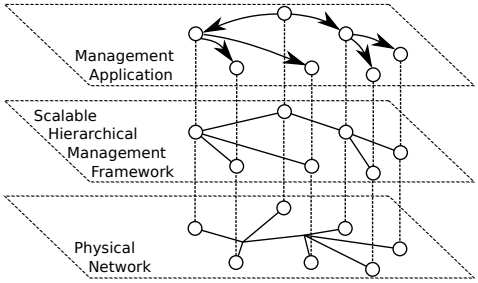


Fig. 1: A hierarchy of nodes is constructed and managed on top of the physical network. The resulting hierarchy is used by a management application that manages the child nodes defined by the hierarchy. This management application can for example execute application placement or context dissemination.

Hierarchical management systems can be used in many contexts, such as context distribution [2], [3], and cloud resource management [4]. The way in which the hierarchy is structured can however impact the performance of a management system, and the management system can impose additional constraints on the hierarchy: the number of nodes managed at every level in the hierarchy may be limited; it can be preferable for the physical location of nodes within a cluster to be close together, especially in federated clouds where communication between nodes in different datacenters must be avoided due to latency and bandwidth limitations; and constraints can limit which nodes may occur together as siblings in the hierarchy.

In this paper, we describe the Scalable Hierarchical Management Framework (SHMF), a framework that can be used to build various hierarchical management applications. The hierarchical framework automatically constructs and maintains a hierarchy, that can then be used by the managing application, essentially managing the structure of the management application. This is illustrated in Figure 1. The structure of the hierarchy is optimized based on the requirements imposed by the application, and can thus vary depending on the application for which it is used. The architecture of the SHMF can be used for various hierarchical management systems. Within this paper, we specifically focus on the hierarchy requirements imposed by cloud application placement [4].

In the next Section, we discuss related work. In Section III we discuss the requirements for a hierarchical management framework, and in Section IV we discuss a formal hierarchy

ISBN 978-3-901882-53-1, 9th CNSM and Workshops ©2013 IFIP

model using these requirements. In Section V we describe an architecture for the hierarchical framework, and subsequently, in Section VI we describe algorithms to manage the hierarchy. We compare the performance of the hierarchical structure with optimal results derived from the formal model in Section VI-D. Finally, we state our conclusions in Section VIII.

II. Related Work

In this paper we study a hierarchical management framework based on the requirements of the application placement problem [5]. Multiple centralized solutions to solve this problem have been proposed [6], [7], [8], [9]. Fully distributed solutions, that lack a global system overview, have also been proposed in literature. These approaches work using peer-to-peer communication [10], [11] and economic approaches where requests are traded between nodes [12], [13]. The development of hierarchical placement algorithms was the focus of our previous work [4], where we concentrated on the design of hierarchical application placement algorithms. In this paper, we however focus on the structure of the management hierarchy itself, and how it can be constructed and managed in a scalable way, rather than on the management algorithms used on top of the hierarchical structure.

Hierarchical approaches are also used for other purposes within cloud and network environments, such as context dissemination. Aggregating this information hierarchically improves scalability [1], and makes it possible to execute varying management tasks, requiring different network overviews, at different hierarchy levels [2], [3]. These works however focus on algorithms and approaches for hierarchical context dissemination, rather than on the structure of the hierarchy itself. We, by contrast, focus specifically on how hierarchies can be constructed within distributed environments.

Our approach has similarities to [14] and [15], where an automatic hierarchical node ordering in peer to peer systems is presented. The hierarchical system in these works is however mainly used to structure object lookup, and every node can occur at multiple levels in the tree. In our framework, by contrast, every node only occurs once as it is either used as a management node, or as an execution node, but not as both. Our approach further differs, as it is aware of additional requirements defined by a managing application impacting the tree structure.

III. Hierarchical Framework Requirements

In general, the architecture of a hierarchical framework must provide two important qualities:

Q1 – Scalability Hierarchies are used to ensure the management system is scalable, but to achieve this the communication between nodes must be limited, and management information must be aggregated in a scalable way.

Q2 – Robustness A disadvantage of a hierarchical approach is that failure of a single hierarchy node causes the management system to become disconnected. Thus, the hierarchy must be robust, and be able to repair and restructure itself when node failures occur.

Additionally, the management application in which the framework is used can also impact the requirements for the hierarchy. Within this paper, we focus on the hierarchy requirements elicited in our previous work [4], where a hierarchical approach for cloud application placement is proposed. We discern four requirements:

H1 – Child Node Limit: The performance of the management algorithm is dependent on the number of child nodes of a node in the hierarchy. Thus, the number of child nodes of any node within the tree should be limited to ensure the execution of the management algorithms does not take too long.

H2 – Execution Node Maximization: The inner nodes of the management hierarchy, referred to as management nodes, are used to manage the environment, while leaf nodes, referred to as execution nodes, are used to execute the cloud applications. The number of management nodes should be minimized, as these nodes result in the management overhead. Conversely, the number of execution nodes should be maximized.

H3 – Sibling Restriction: The management algorithm can discern two tasks: managing a collection of execution nodes, and managing a collection of management nodes. The former task corresponds to a scenario where a centralized management algorithm is executed in a small section of the management system. The latter task however differs, as there, the requests are divided between management nodes. While in some cases the same algorithms can be used for both tasks, the algorithms behave differently, and in some instances it can be desirable to execute different management algorithms for these tasks. Because of these considerations, it is necessary to ensure that a node either manages a cluster containing *only* execution nodes, or manages one containing *only* management nodes.

H4 – Execution Cluster Distance Minimization: The goal of a hierarchical approach is to execute requests at lower levels in the hierarchy, ensuring higher level management nodes do not need to be aware of these requests. Because of this, it is desirable for nodes that are close together within the hierarchy to also be close together physically, as this reduces the network overhead. Thus, the distance between nodes in the clusters created by grouping the execution nodes together based on their parent node (thus ensuring siblings are placed in the same clusters) should be minimized.

IV. Hierarchical Model

The management hierarchy is constructed using a collection of computation nodes N. A binary decision variable X_n^p determines whether a given node n is a child of a node p within the hierarchy: p is a parent of n iff $X_n^p = 1$. The distance between different nodes is stored in the distance matrix D. D_{uv} represents the distance between nodes u and v.

First, we define a set of constraints ensuring the nodes form a hierarchy. For this, it is important to ensure that every node has at most one parent. This is expressed in Equation (1).

$$\forall n \in N : \sum_{p \in N} X_n^p \leq 1 \tag{1}$$

Additionally, every node, except for the root, must have exactly one parent. This implies that, in total, there must be

exactly $|N|-1$ edges between nodes, which is enforced by Equation (2).

$$|N|-1=\sum_{(n,p)\in N} X_n^p \tag{2}$$

Finally, a constraint is needed to prevent cycles from occurring. We do this by, for every node n, determining the height of the node H^n. Logically, the height of a node must be higher than the height of each of its child nodes. This is expressed in Equation (3) and Equation (4), where the constraint resolves to $H^p \geq H^l+1$ if the node p is a parent of node n (thus if $X_l^p = 1$), while otherwise no constraint is enforced as the height cannot be higher than $|N|$. This approach prevents cycles from occurring, as each parent node has a higher height than each of its children, and a cycle would imply that a node must have a higher height than itself.

$$\forall (p,n)\in N^2 : (|N|+1)\times(1-X_l^p)+H^p \geq H^n+1 \tag{3}$$

$$\forall p\in N : H^p \in [0,|N|] \tag{4}$$

A limit to the number of child nodes, b, is chosen as an input for the model. No tree node may have more than b child nodes. This is expressed in Equation (5).

$$\forall p\in N : \sum_{n\in N} X_n^p \leq b \tag{5}$$

An objective of the model is to maximize the number of leaf nodes, as these are the nodes that are used for the actual execution of cloud applications, while the inner nodes of the tree are used to manage the infrastructure. Thus, it is important to determine whether a node is a leaf node, for which we use the binary variable L^p. The value of L^p is determined using two equations. If a node p has no children, it is a leaf; this is expressed in Equation (6). Conversely, if a node p has a child node, it is not a leaf. The latter is expressed in Equation (7).

$$\forall p\in N : 1-L^p \leq \sum_{n\in N} X_n^p \tag{6}$$

$$\forall p\in N : \forall n\in N : 1-L^p \geq X_n^p \tag{7}$$

As specified, it is possible to determine whether a two nodes occur in a parent-child relationship. It is however not yet possible to determine whether two separate nodes are siblings, which is necessary to determine the quality of a group and to ensure all neighbors of leaf nodes are leaves as well. For this, we introduce the decision variable X_{nm}^p, which takes on value 1 if both nodes n and m are children of p, and which takes on value 0 otherwise. Equation (8) expresses that if $X_{nm}^p = 1$, X_n^p and X_m^p must also both equal 1. Conversely, Equation (9) expresses the opposite, and ensures that if $X_n^p = X_m^p = 1$, X_{nm}^p must also equal 1:

$$\forall (p,n,m)\in N^3 : 2\times X_{nm}^p \leq X_n^p + X_m^p \tag{8}$$

$$\forall (p,n,m)\in N^3 : X_n^p + X_m^p - 1 \leq X_{nm}^p \tag{9}$$

This additional decision variable makes it possible to ensure that all neighbors of a leaf node are also themselves leaf nodes. Equation (10) is used to express that if a node n is both a leaf and a sibling of a node m, the node m itself must also be a leaf node.

$$\forall (p,n,m)\in N^3 : L^n + X_{nm}^p - 1 \leq L^m \tag{10}$$

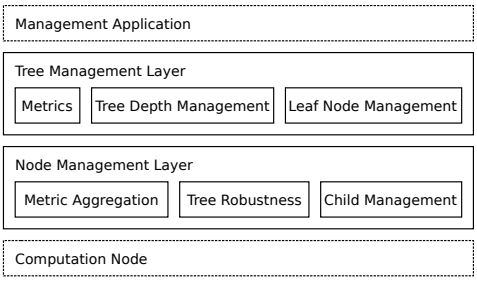


Fig. 2: An overview of the SHMF architecture.

We determine the quality of a cluster as the maximum distance between any of its child nodes. This metric is only relevant for leaf nodes, as the quality of the complete tree is determined by the quality of the leaf clusters. For every node p, the leaf cluster quality Q_n of its children is determined as shown in Equation (11). If p is a leaf node, the equation is reduced to $Q_p \geq X_{nm}^p \times D_{nm}$, thus if both nodes are a child of p, Q_p is at least the distance between these nodes. If p is no leaf node, the maximum distance D^{max} is subtracted, ensuring no constraint on Q_p is added.

$$\begin{gathered}\forall p\in N : Q_p \geq 0 \\ \forall (p,n,m)\in N^3 : Q_p \geq X_{nm}^p \times D_{nm} - (1-L^p)\times D^{max} \\ D^{max} = \max_{(n,m)\in N^2} D_{nm}\end{gathered} \tag{11}$$

Based on the requirements discussed in Section III, two optimizations must be executed. First, the number of leaf nodes must be maximized, ensuring there is a maximum number of executing servers. This objective is shown in Equation (12).

$$\max \sum_{n\in N} L^n \tag{12}$$

This first optimization results in a maximum number of leaf nodes L^{max}, which can be used to add an additional constraint, shown in Equation (13), to the model.

$$L^{max} \leq \sum_{n\in N} L^n \tag{13}$$

The second optimization is to minimize the distances between leaves in the tree. This is expressed in Equation (14).

$$\min \sum_{p\in N} Q_p \tag{14}$$

V. SHMF Architecture

The architecture of SHMF, shown in Figure 2, consists of four layers. On the computation node, a node management layer is deployed that manages the node. This layer is responsible for maintaining a relationship with the node's children and aggregating management information. The tree management layer manages the structure of the hierarchy. Finally, a management application makes use of the constructed hierarchy to execute management tasks such as cloud application placement or context distribution.

A. Node Management Layer

The node management layer is responsible for controlling the relationship between a node within the hierarchy and its child nodes. To the layers above, it abstracts the physical node and manages communications. The node management layer has three functionalities:

1) The node management layer maintains a collection of child nodes, and manages these nodes.

2) It provides functionality and infrastructure to aggregate management information from its child nodes: every node views only metrics it aggregates from its child nodes, and does not require any other management information. This information aggregation can be achieved in a scalable way, e.g. by managing all child nodes using a P2P structure, limiting communication to and from the management node. This design limits the information every node requires, and fulfills the scalability requirement **Q1**.

3) Additionally, the node management layer is responsible for ensuring the tree robustness: when a node failure is detected, the node automatically re-adds itself to a known existing node of the hierarchy. It is possible to make use of a highly-connected unstructured management overlay such as CYCLON [16], to further enhance the reliability of the management system. Using such an overlay, a management node knows a large collection of other tree nodes, ensuring it would take a catastrophic failure for the management system to become disconnected. This system is used to fulfill requirement **Q2**, the reliability of the management system.

The implementation of the node management layer is not the focus of this paper: its implementation is simulated to ensure large scale evaluations can be executed using a single machine.

B. Tree Management Layer

The tree management layer runs on top of the hierarchy provided by the node management layer. This layer issues commands to the node management layer using logical references to other nodes, and optimizes the hierarchy's structure. The layer defines a set of metrics that are used for management purposes that are aggregated by the node management layer, and executes two management strategies using this information:

1) A tree depth management strategy is executed to address hierarchy requirements **H1** and **H2**: the number of child nodes of limited based on a limiting branching factor B, and a node's child nodes are packed as tightly as possible, maximizing the number of child nodes within the subtree.

2) A leaf node management strategy that specifically focuses on hierarchy requirement **H3**: this management strategy ensures a node either manages management nodes, or execution nodes, but never both.

Both management strategies consider the average cluster distance of the resulting system, requirement **H4**, when taking decisions during their execution.

TABLE I: The management strategies and the required management metrics.

	Tree Height Management	Leaf Node Management
Node Count	x	x
Weighted Medoid	x	
Bad Node		x
Accept Node		x
Height	x	x
Potential Node Count		x

VI. SHMF Management Algorithms

The SHMF tree self-organizes using multiple algorithms discussed within this section. These management algorithms function using locally available information that is aggregated by the node management layer, referred to as metrics, and respond to changes in these metrics. Table I shows which metrics are used by each management algorithm. In this section we first discuss the various management metrics used by the algorithms. Subsequently the tree height and leaf node management algorithms are explained. Finally a tree construction algorithm is described.

A. Management Metrics

The management system aggregates four metrics, that are based on aggregate values computed from the node's child nodes. At every node, the value of each of these metrics is stored. Aggregating these metrics is a task of the node management layer in the system architecture.

1) Node Count: The most important metric used in the structuring of the management tree, is the node count of the tree. For every node n this value can be easily computed using Equation (15), where $\mathrm{children}(n)$ is the set of child nodes of the node n, and $\mathrm{nc}(n)$ is the value node count metric of the node n.

$$\mathrm{nc}(n) = 1 + \sum_{c \in \mathrm{children}(n)} \mathrm{nc}(c) \tag{15}$$

2) Weighted Cluster Medoid: The weighted cluster medoid aggregation is used to calculate the medoid of a given cluster. The cluster medoid is the element of the cluster that best characterizes it. This medoid can be determined by finding the node for which the distances to all other nodes is the smallest. For higher nodes in the hierarchy, the number of nodes in the subtree become increasingly large. As it is impossible to accurately compute this medoid value in a scalable way, we estimate the medoid of a node based on the medoids of its child values, where these medoids are weighted based on the number of nodes in the subtree of every child node. The formal computation of the medoid value of a node n, $\mathrm{med}(n)$, is shown in Equation (19). This formula determines the best medoid m, that results in the lowest distance, quantified as $\mathrm{mq}(n, m)$, out of all of the possible medoid values, $\mathrm{possible}(n)$. The medoid value of a node n can either be the node itself, or the medoid of any of its child nodes; this is expressed in Equation (16). The quality of a given medoid m for a given node n is shown in Equation (18), and is dependent on the number of nodes the

medoid m represents. Equation (17) shows how this frequency is computed.

$$\text{possible}(n) = \{n\} \cup \left(\bigcup_{c \in \text{children}(n)} \text{med}(n) \right) \tag{16}$$

$$\text{eq}(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases}$$

$$\text{freq}(n,m) = \text{eq}(m,n) + \sum_{c \in \text{children}(n)} \text{eq}(\text{med}(c), m) \times \text{nc}(c) \tag{17}$$

$$\text{mq}(n,m) = \sum_{m' \in \text{possible}(n)} \|m\ m'\| \times \text{freq}(n,m') \tag{18}$$

$$\text{med}(n) = \underset{m \in \text{possible}(n)}{\arg\min}\ \text{mq}(n,m) \tag{19}$$

3) Bad Node Aggregation: The bad node aggregation contains a collection of badly fitting tree nodes, which is used within the leaf node management. The tree management layer can assign a node to this aggregation, notifying higher level tree nodes that the quality of the tree will increase if the node is moved. The value of this aggregation is calculated as shown in Equation (20), where N either contains a single node, the bad node specified by the management system at node n, or is the empty set if no bad node is selected.

$$\text{badNodes}(n) = N \cup \left(\bigcup_{c \in \text{children}(n)} \text{badNodes}(c) \right) \tag{20}$$

4) Accept Node Aggregation: The accept node aggregation has the opposite function of the bad node aggregation: it is used to determine places within the management hierarchy where it would be preferable to add additional nodes. Its formulation is similar to that of the bad node aggregation, but instead it aggregates parent nodes that do not have b child nodes.

Using the values of the node count metric, two additional metrics can be computed:

5) Node Height: The node height metric determines the height of a node in a tree. Traditionally, this value is calculated by determining the height of all child nodes, choosing the maximum value, and increasing it by one; a leaf node is assigned height 0. We however use an alternative approach to determine the height of a node n: based on the number of child nodes $\text{nc}(n)$ contained in the subtree, and the branching factor b of the tree, we can determine the height a node should have. This height is determined using Equation (21).

$$\text{height}(n) = \lceil \log_b(b-1) \times \text{nc}(n) + 1 \rceil \tag{21}$$

The advantage of this approach is that, the height of a node will change less frequently and be less dependent on the actual height of child nodes. This in turn makes it easier to localize height changes within the tree: if a node n has height $\text{height}(n)$, while its children have a height value that is too low, this can then be solved by node n. Otherwise, the height of node n would have been lower, making the parent of n responsible for fixing the imbalance.

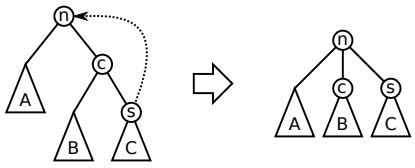


Fig. 3: The takeChild operation. This operation is invoked if $\text{height}(n) = \text{height}(c)$ for a child node c. A grandchild node s is moved, and added as a child of n, ensuring the node count $\text{nc}(c)$ decreases, which in turn can decrease $\text{height}(c)$.

6) Potential Node Count: This metric determines the maximum number of nodes that a subtree can possibly have, given its height. This is calculated by determining the total number of nodes in a complete tree of the given height, and can be computed using Equation (22).

$$\text{pnc}(h) = \frac{b^{h-1} - 1}{b - 1}$$
$$\text{pnc}(n) = \text{pnc}(\text{height}(h)) \tag{22}$$

B. Tree Height Management Algorithm

The tree height management strategy is responsible for managing the height of a node's child nodes. This strategy guarantees requirement **H1**, the number of child nodes of a node is less then or equal to b, and tries to maximally fill every subtree, which corresponds to requirement **H2**.

The tree height management strategy operates by observing the height of its child nodes, and invoking an update operation when any changes occur. This update can invoke two operations: (1) if one or more of the child nodes are as deep as this node, the takeChild function is called, otherwise (2) the mergeChildren function is invoked.

The $\text{takeChild}(n)$ operation is invoked when one or more of the child nodes of n has the same height as n. These child nodes are contained in the set C. The method takes a single child node s_c of each of these child nodes $c \in C$, and directly attaches them as a child of the node n. This decreases the number of nodes in the subtree of every node $c \in C$, which in turn can decrease the height of these nodes. This operation is illustrated in Figure 3.

The $\text{mergeChildren}(n)$ operation is invoked when all child nodes have a low enough height value. It attempts to merge the child nodes of n, ensuring the number of child nodes of n is less or equal to the branching factor b, and ensuring every subtree with root a child of n has as many child nodes as possible considering its potential node count.

To achieve this, a modified best-fit bin-packing algorithm is executed, where $\text{children}(n)$ are grouped into bins of size $\text{pnc}(\text{height}(n)-1)$. The size of a child $c \in \text{children}(n)$ in the bin-packing algorithm equals its node count, $\text{nc}(c)$. The bin-packing algorithm uses a fixed number of bins: at the start of the algorithm, b bins are created, and during the execution of the algorithm no additional bins can be created. As no additional bins are created when needed, it is possible that some child nodes will not fit in a bin; these nodes are grouped

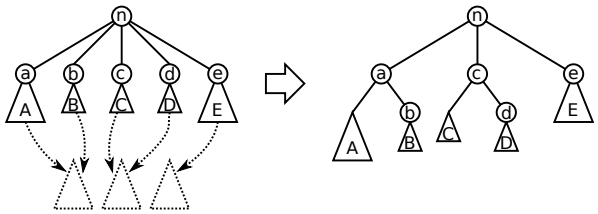


Fig. 4: An example invocation of the $\mathrm{mergeChildren}$ operation. Child nodes of a node n are grouped using a modified bin-packing algorithm. The nodes within a group are then merged into a single subtree, where nodes with a lower height are added as child nodes to nodes with the highest height.

in the failed set F. Child nodes with the highest $\mathrm{nc}(c)$ are placed first, as they are more difficult to place. A child node is placed in the bin that results in the lowest distance between nodes like in the weighted medoid calculation described in the previous section.

The bin-packed child nodes are then merged into a single subtree, by adding the nodes with a lower height value as a child node of the node with the highest height value. Nodes f, contained in the set F, that were not fitted into bins, are then processed separately: the child nodes $c \in \mathrm{children}(f)$ of these nodes are directly added to the node n, while the node f itself is added to the best-fitting child node of n. The $\mathrm{mergeChildren}$ operation is illustrated in Figure 4.

After executing the $\mathrm{mergeChildren}(n)$ operation, there will either be no failed nodes, which ensures $|\mathrm{children}(n)| < b$, or some nodes will fail. When nodes fail, the failing subtrees are split into smaller trees, ensuring they have a smaller granularity, and making it easier to fit them into bins in a subsequent invocation of $\mathrm{mergeChildren}$.

Both the $\mathrm{takeChild}$ and $\mathrm{mergeChildren}$ operations partially improve the structure of the tree. If either of the operations changes the subtree, this changes either the number of children of the tree or the height of nodes, which causes a new update operation to be invoked. In this subsequent operations, the structure of the tree will then be further improved. This repeats until the height of all child nodes is less than the height of the node itself, and the number of child nodes of the node is less than or equal to b.

C. Leaf Node Management Algorithm

The second management strategy is the leaf node management strategy. This strategy focuses on the requirement **H3** of the hierarchy specified in Section III: a node should either manage only other tree nodes, or it should only manage leaf nodes. This is achieved making use of two operations: an update operation that is executed when the height of a node's children changes, and a bad node aggregation system. The former is used to try to fix an imbalance at the local level, while the bad node aggregation system is used in situations where an imbalance can not be resolved locally and must be fixed by a node at a higher level in the hierarchy.

When an update is invoked, the height of child nodes is evaluated. If the node has child nodes with height 0, and other child nodes with height > 0, the structure of the tree is modified to resolve this. We assume in this section that the management strategy is executed on node n, and that its child nodes are partitioned into two sets: L, containing leaf nodes, and $\overline{L}$ containing other nodes. Three possible operations can be executed:

1) It is often possible to move the nodes in L, and add them as children of the nodes in $\overline{L}$. This should only be done if there is sufficient space in these subtrees to move all of the nodes in L. This is the case if the potential node count of the child nodes, calculated based on the height of the nodes, is higher than the actual node count. Formally, this is done if $\sum_{c \in \overline{L}}(\mathrm{pnc}(c) - \mathrm{nc}(c)) \geq |L|$.

2) If there are multiple leaf nodes these nodes can be grouped together: a single node r is chosen from L, and all other nodes in L are added as a child of r.

3) If none of the above are applicable, there is only a single problematic leaf node, and it can not be moved to one of its siblings. This problem cannot be resolved at this hierarchy level, therefore, the leaf node is stored in the management node's bad node metric. Through the metric aggregation mechanism parent and grandparent nodes of n are notified, and they can then resolve the problem.

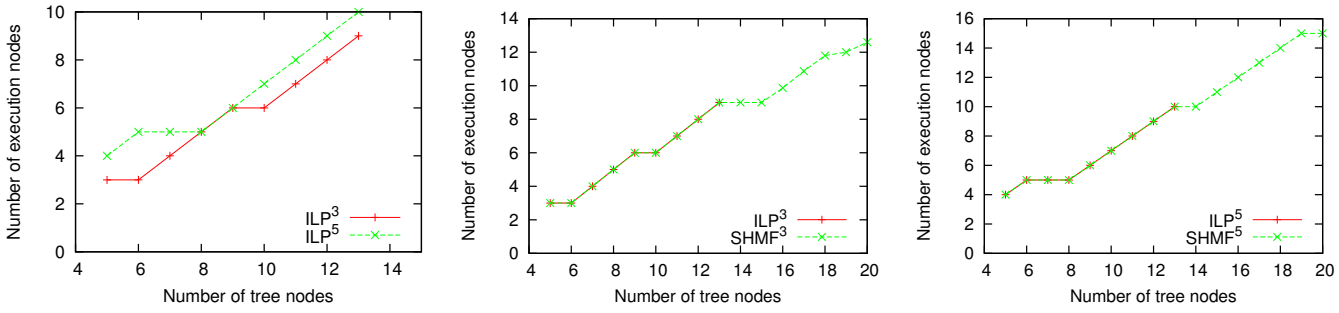
The accept node metric and the bad node metric are used by the leaf node management strategy to move problematic imbalances that can not be solved at this node level, but that must be fixed at a higher level in the hierarchy. The bad node metric makes it possible to specify leaf nodes that are positioned badly within the tree, and that should be moved. The accept node aggregation conversely notifies that a node does not have the maximum amount of child nodes B, and can accept additional child nodes. The leaf node management strategies observes both aggregations, and matches bad nodes to accept nodes, moving bad nodes to other locations in the hierarchy where there is room for them. Additionally, this strategy uses a the bad node aggregation to offer management nodes with only a single child node, allowing for them to be used as execution nodes at a different position in the hierarchy.

D. Tree Construction Algorithm

The various management algorithms continuously monitor changes in the metrics. A tree can be constructed by selecting a single node that functions as the hierarchy root, and iteratively adding nodes to any tree node. While the tree construction can be random, where a node chooses a random node of the tree to attach itself to, we use an approach where a node chooses the nearest node within the hierarchy to attach itself to. This approach is realistic for large environments, where a node attaches itself to the nearest node it can find that is already present within the hierarchy. By adding nodes to the hierarchy, the value of metrics changes, causing the various management strategies to be activated.

VII. Evaluation Results

We evaluate both the formal model discussed in Section IV, and the architecture and algorithms specified in Sections V and VI. The formal model from Section IV is implemented



(a) Comparison of ILP algorithms. (b) Comparison for $b = 3$. (c) Comparison for $b = 5$.

Fig. 5: The number of execution nodes achieved by the SHMF and ILP algorithms for varying node counts and branching factors.

as an Integer Linear Programming (ILP) model using the CPLEX [17] ILP solver, resulting in an optimal tree structure. We refer to this algorithm as ILP^b, where b is the branching factor used in the algorithm. Similarly, the SHMF^b algorithm constructs a tree as discussed in Section VI-D and uses the various presented algorithms to restructure itself, taking into account a branching factor b.

The input of the algorithms is entirely defined by the number of nodes and the distance matrix D. For every data point in the graphs, 15 evaluations were executed using varying D matrices. The values of every D are uniformly chosen in the range $[0, 10]$. For the purposes of our evaluation, the unit used in D is not important: the distance matrix can be rescaled to represent e.g. the hop count between nodes in a datacenter, or the latency of communication between compute nodes.

We first evaluate the achieved node count of the SHMF algorithm, comparing it to the optimal node count determined by the ILP algorithm. The ILP algorithm scales badly, and can only be used for very small node counts and branching factors. In Figure 5a, we compare the execution node count of the ILP algorithm with branching factors 3 and 5. We observe that, as the number of tree nodes increases, the number of execution nodes generally increases as well. For some node counts, the number of execution nodes does not increase: at this point, the additional nodes are used as management nodes due to the restrictions of the hierarchy.

When the number of execution nodes of the ILP algorithm is compared to that achieved by the SHMF algorithm, as shown in Figures 5b and 5c, we observe that the number of execution nodes similarly increases as the number of nodes increases. In all of the evaluated cases, where the ILP algorithm could be executed, the optimal execution node count was achieved.

The SHMF algorithm however scales much better, as it can easily be executed on a single node for large node counts, as evidenced in Figure 6, where the tree algorithm is compared for larger node counts. In the Figure, the number of management nodes used for various SHMF algorithms is compared. The number of tree nodes used by the SHMF algorithm increases linerarly as the number of tree nodes increases. The higher the branching factor b, the less management nodes are needed within the hierarchy, thus decreasing the management overhead. With a branching factor of 50, a management tree

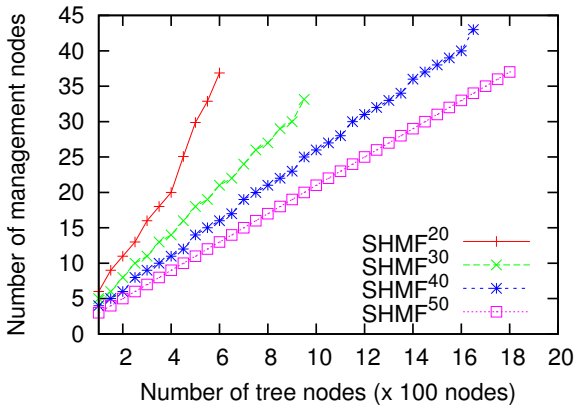
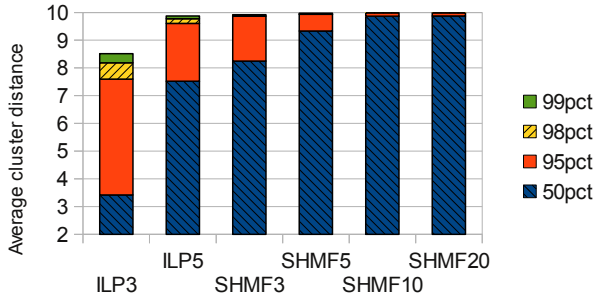


Fig. 6: Evaluation of the number of management nodes for large management trees.

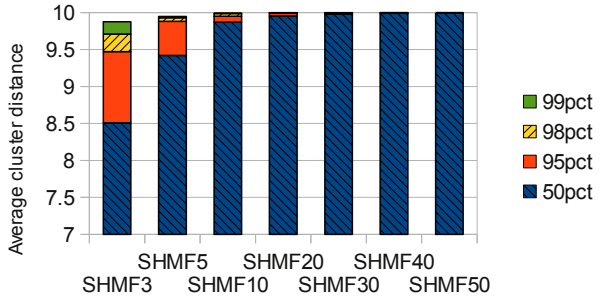
containing 1500 nodes requires only ± 30 management nodes. Note that larger branching factors require less tree reorganisations, making it possible to evaluate higher node counts.

The quality of the resulting hierarchies is evaluated in Figures 7a and 7b, where the average cluster distance between the different algorithms and branching factors is compared. A first observation is that the quality of clusters generally decreases as the branching factor increases. This is to be expected, as when there are more nodes in a cluster, it is more difficult to limit the number of nodes per cluster. As shown in Figure 7a, the ILP algorithm achieves higher quality results than the SHMF algorithm but, as noted previously, this algorithm can only function until a node count of 14.

When the full dataset is taken into account, as shown in Figure 7b, we observe that the quality achieved by the SHMF algorithms with higher branching factors improves slightly compared to the smaller data set. This can be expected, as in a larger set there are more nodes amongst which the cluster can be chosen, whereas in the smaller subset, this choice is much more limited. In the data set with smaller node counts, SHMF^{20} will, for example always result in a single cluster containing all nodes except for one which is chosen as the root; making the average cluster distance of the tree completely dependent on the distances in D.



(a) Average cluster distance with node count ≤ 14



(b) Average cluster distance with node count ≤ 1500

Fig. 7: The distribution of the average distance between nodes in a cluster in percentiles.

VIII. Conclusion

In this paper, we described SHMF, a scalable approach for managing a hierarchical cloud management system. We discussed the framework requirements, which were elicited based on a hierarchical cloud application placement application. A model, incorporating these requirements, and capable of maximizing the leaf cluster distance of the hierarchical structure was described. Additionally, the architecture and algorithms for a hierarchical management framework were also presented. Subsequently, the trees constructed using the optimal algorithm were compared to those created using SHMF. We found that the SHMF algorithm scales much better, as it functions using limited information and uses simple management algorithms, making it much more suitable for large scale use cases. Additionally, while the optimal algorithm outperforms SHMF when it comes to the cluster distance of the hierarchical structure, the SHMF algorithm achieved the optimal number of execution nodes for all cases where the optimal algorithm could still be executed.

Acknowledgment

Hendrik Moens is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT). This research is partly funded by the iMinds PUMA [18] project, and was carried out using the Stevin Supercomputer Infrastructure at Ghent University, funded by Ghent University, the Hercules Foundation and the Flemish Government – department EWI.

References

[1] J. Famaey, S. Latré, J. Strassner, and F. De Turck, "A hierarchical approach toautonomic network management," *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*, pp. 225–232, Apr. 2010.

[2] ——, "A Hierarchical Context Dissemination Framework for Managing Federated Clouds," *Journal of Communications and Networks*, vol. 13, no. 6, pp. 567–583, 2011.

[3] ——, "Semantic Context Dissemination and Service Matchmaking in Future Network Management," *International Journal of Network Management*, vol. 22, no. 4, pp. 285–310, 2012.

[4] H. Moens, J. Famaey, S. Latré, B. Dhoedt, and F. De Turck, "Design and Evaluation of a Hierarchical Application Placement Algorithm in Large Scale Clouds," in *Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, 2011, pp. 137–144.

[5] J. Rolia, A. Andrzejak, and M. Arlitt, "Automating enterprise application placement in resource utilities," in *Self-Managing Distributed Systems: 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2003)*. Springer, 2004, pp. 118–129.

[6] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers," in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 331–340.

[7] T. Kimbrel, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic application placement under service and memory constraints," in *Proceedings of the 4th international conference on Experimental and Efficient Algorithms*, Apr. 2005, pp. 391–402.

[8] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi, "Dynamic placement for clustered web applications," in *Proceedings of the 15th international conference on World Wide Web*. ACM, 2006, pp. 595–604.

[9] D. Carrera, M. Steinder, I. Whalley, J. Torres, and E. Ayguadé, "Utility-based placement of dynamic web applications with fairness goals," in *Proceedings of the 11th Network Operations and Management Symposium (NOMS 2008)*. IEEE, 2008, pp. 9–16.

[10] C. Adam and R. Stadler, "Service Middleware for Self-Managing Large-Scale Systems," *IEEE Transactions on Network and Service Management*, vol. 4, no. 3, pp. 50–64, Dec. 2007.

[11] F. Wuhib, R. Stadler, and M. Spreitzer, "Gossip-based Resource Management for Cloud Environments," in *Proceedings of the 6th International Conference on Network and Service Management (CNSM 2010)*, 2010, pp. 1–8.

[12] Y. Li, F.-H. Chen, X. Sun, M.-H. Zhou, W.-P. Jiao, D.-G. Cao, and H. Mei, "Self-Adaptive Resource Management for Large-Scale Shared Clusters," *Science And Technology*, vol. 25, no. 2009, pp. 945–957, 2010.

[13] C. Low, "Decentralised Application Placement," *Future Generation Computer Systems*, vol. 21, no. 2, pp. 281–290, 2005.

[14] B. Hudzia, M.-T. Kechadi, and A. Ottewill, "TreeP: A Tree Based P2P Network Architecture," *2005 IEEE International Conference on Cluster Computing*, pp. 1–15, Sep. 2005.

[15] E. Edi, T. Kechadi, and R. McNulty, "TreeP: A Self-reconfigurable Topology for Unstructured P2P Systems," in *Applied Parallel Computing. State of the Art in Scientific Computing*, ser. Lecture Notes in Computer Science, E. Kå gström, Boand Elmroth, J. Dongarra, and J. Waśniewski, Eds. Springer Berlin Heidelberg, 2007, vol. 4699, pp. 1136–1146.

[16] S. Voulgaris, D. Gavidia, and M. Steen, "CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, Jun. 2005.

[17] (2013) IBM ILOG CPLEX 12.4. [Online]. Available: http://www-01.ibm.com/software/integration/optimization/cplex-optimizer

[18] (2013) PUMA: permissions, user management and availability for multi-tenant saas applications. [Online]. Available: http://www.iminds.be/en/research/overview-projects/p/detail/puma-2