Autonomous Multi-Dimensional Slicing for Large-Scale Distributed Systems

Mathieu Pasquet[1⋆], Francisco Maia[2], Etienne Rivière[3], and Valerio Schiavoni[3]

[1] École Normale Supérieure Rennes, France. mathieu.pasquet@ens-rennes.fr
[2] High-Assurance Software Laboratory, INESC TEC & University of Minho, Portugal. fmaia@di.uminho.pt
[3] Université de Neuchâtel, Switzerland. first.last@unine.ch

Abstract. Slicing is a distributed systems primitive that allows to autonomously partition a large set of nodes based on node-local attributes. Slicing is decisive for automatically provisioning system resources for different services, based on their requirements or importance. One of the main limitations of existing slicing protocols is that only single dimension attributes are considered for partitioning. In practical settings, it is often necessary to consider best compromises for an ensemble of metrics.
In this paper we propose an extension of the slicing primitive that allows multi-attribute distributed systems slicing . Our protocol employs a gossip-based approach that does not require centralized knowledge and allows self-organization. It leverages the notion of domination between nodes, forming a partial order between multi-dimensional points, in a similar way to SkyLine queries for databases. We evaluate and demonstrate the interest of our approach using large-scale simulations.

Keywords: Slicing, Self-organization, Skyline, Gossip-based protocols

1 Introduction

Very large-scale heterogeneous distributed systems will be commonplace with the advent of connected objects, the internet-of-things and in general with the massive increase in the number of machines connected through multiple networks. Such very large systems can no longer be operated using the system services and support mechanisms that were designed and implemented for moderate to small scale distributed systems. In particular, system-level primitives and services, that were once supported by centralized entities or by the close collaboration of a few synchronized nodes, may not be adequate or even possible to use in these scenarios. The characteristics of large-scale heterogeneous systems call instead for fully decentralized and autonomous protocols. Such protocols can provide system-wide fundamental system services but do not require centralized components or complex synchronization between nodes. Self-organization is another key requirement. It denotes the ability of a system or service to seamlessly adapt to

⋆ Work done while interning at Universit de Neuchâtel.

dynamic conditions, such as nodes heterogeneity, dynamic membership, or faults. It allows always ending up in a regular operating state, regardless of the starting conditions or perturbations imposed to the system.

Self-organizing, autonomous distributed protocols and services are typically implemented using the *gossip-based* paradigm. Gossip-based protocols rely on periodic exchange of information between pairs of peers and appropriate node-local decisions. They allow implementing a variety of system services based on overlay networks, such as membership management [16], dissemination [7, 19], structure and overlays management [15, 24], or key-based routing [18, 20].

An important class of services for large-scale heterogeneous distributed systems is *resource provisioning*. A system might need to support multiple services and applications, each with different requirements in terms of computational capabilities, network requirements, or criticality (i.e., the need for the service to be dependable and available). Resource provisioning allows determining, for each service or application, which resources should be used. In particular, when services are implemented through distributed protocols running on a subset of all the nodes of the system, provisioning can be generalized to the action of splitting the entire set of nodes in subsets, one for each of the service or application.

For instance, in a system that supports indexing and dependable storage atop highly volatile nodes [18], a small set of nodes –typically the most stable ones– can be dedicated to the routing service that implements the indexing layer while the remaining ones handle best effort replication of data items. The selection of the most stable nodes can be based on their uptime. As demonstrated in [4] based on measurements on a real large peer-to-peer system, uptime is correlated with stability. Nodes with large uptimes tend to stay online for a longer time than nodes with a small uptime, regardless of their other characteristics. Another example is that of a system that supports a BitTorrent-like dissemination and download service. The subset of the most stable node could be used for supporting a self-organizing distributed hash table (DHT) [15, 20, 24], nodes with the largest available bandwidth could be used as helpers for others to download faster [9], while the remaining ones only act as clients.

1.1 Distributed Slicing

The prominent implementation of resource provisioning for very large scale systems is *distributed slicing* [8, 11, 13, 17]. It allows splitting a large set of nodes, based on discrete representations of their characteristics, in the form of node-local *attributes*. Slicing allows forming groups of nodes with increasing attribute values. A *slicing schema*, known by all nodes, indicate the relative size of each of the slices with respect to the total size of the system. It is important to note that the actual total size of the system does not need to be known by the nodes themselves. As an example, say that the considered characteristic of nodes is their available disk space, as exemplified on the top of Figure 1. Nodes all know the slicing schema $S = \{50\%, 30\%, 20\%\}$. A slicing protocol must create three slices of size 50%, 30% and 20% of the total size of the system, e.g., 5,000, 3,000 and 2,000 nodes for a network of 10,000 nodes, and 7, 4 and 3 nodes for the small

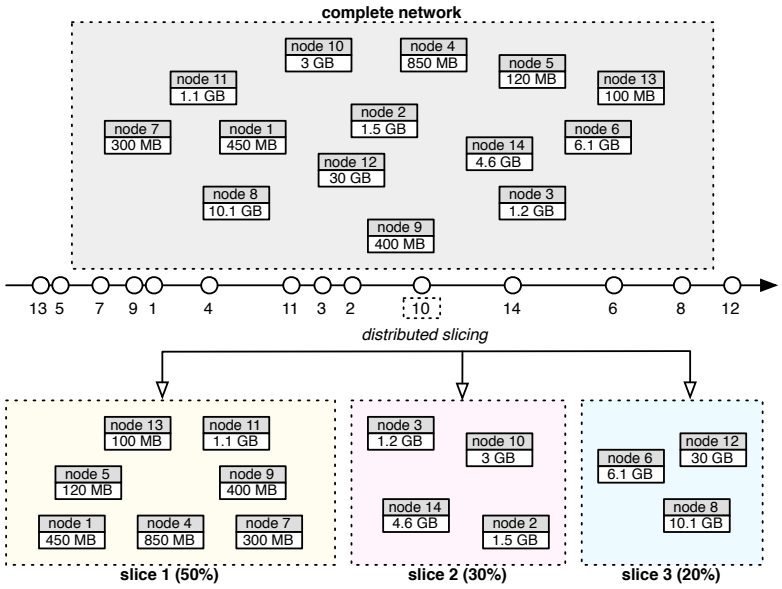

Fig. 1. Basic principle of slicing with attributes of a single dimension (available disk).

network of Figure 1. Slices correspond to group of nodes of increasing attribute values. Nodes in the first slice all have attributes that are lower than those of the second and third slices, and similarly between the second and the third slice.

Slicing neither requires a centralized view of the system, nor it requires assumptions about the distribution of attribute values. Each node must estimate the position its attribute would have in the sorted list of all attributes, if such a list was materialized. This position is typically obtained by estimating the proportion of nodes in the system that have lower values for the attribute and the proportion of nodes that, instead, have higher values for the attribute. When this information is available, it is then straightforward for a node, based on the slicing schema, to determine the slice it belongs to. For instance, node 10 is able to determine that it belongs to the second slice of the schema knowing that there are 9 nodes with lower values, and 4 nodes with higher values. Several variants of protocols allow to implement this basic principle. We give further details of one of these protocols, *sort ranking*, in Section 2.

1.2 Problem definition

The major drawback of slicing for node provisioning in large-scale network is that it intrinsically only supports a single dimension for attributes. It is possible to form slices based on the available disk space, or the network connectivity, or CPU power, but considering a single of these characteristics at a time only. It is often desirable to be able to provision nodes based on a combination of metrics,

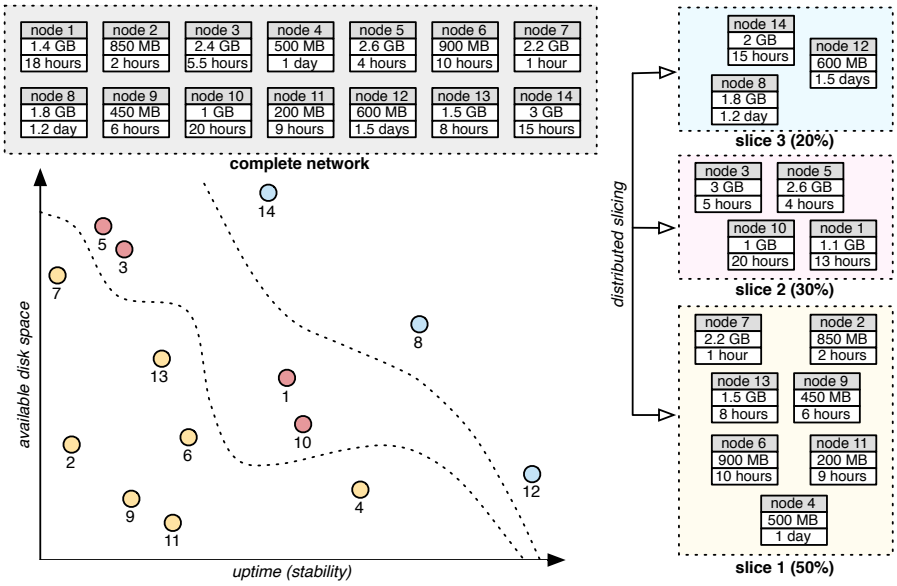

Fig. 2. Principle of the proposed multi-dimensional slicing with two dimensions.

in order to form slices that represent classes of lower to better *compromises* for a set of characteristics. This would allow for instance to provision a large-scale network with a set of super-nodes with the best compromise in terms of available bandwidth and processing capacity, for serving as storage nodes or for participating to a backbone indexing overlay, and use nodes with less interesting characteristics for less demanding tasks, such as dissemination [7] or monitoring [14]. Other examples of compromises involve the node stability as measured by their uptime and processing capacity, in order to form a support overlay composed only of the peers that have both a good capacity, and whose stability ensures good resilience of the structure.

We propose and evaluate in this paper a novel approach to support slicing based on multiple attributes. Our main objective is to form slices that group nodes in classes of quality of the compromises between their different characteristics. This should not require knowing the range of definition for any of the dimensions, which can be practically unbounded. Multi-dimensional slicing should also be independent of the distribution of values for each dimension. Figure 2 illustrates the principle of the considered problem from a high-level perspective. It presents a set of nodes with attributes along two dimensions, the available disk space and the uptime. The latter links to expected stability [4]. Nodes form three slices under the same schema as for Figure 1 over a single dimension, $S = \{50\%, 30\%, 20\%\}$. These slices should represent classes of increasingly better compromises.

We see for instance that node 8 presents both a high uptime and a large amount of available disk space. It is thus expected to be stable, and has enough capacity. On the other hand, node 12 has moderate available disk space but

is the most ancient node, so supposedly the stablest. Taking into account this heterogeneity, the first slice will include nodes such as node 8 or 12 that exhibit the best attribute compromises. Alongside, nodes in the second slice shall represent intermediate compromises among the set of available nodes, while nodes with either small storage space for their stability or small stability for their storage space, are left to the first slice.

Our contribution is based on the determination of *domination relations* between nodes. This notion has been applied to data points in the context of databases for the definition of the SkyLine query [12]. The domination relation between nodes allow comparing compromises between nodes characteristics, and allows nodes to estimate their position in a global domination partial order that would contain all nodes if materialized. This position is used, as for the uni-dimensional slicing, to determine the slice a node belongs to.

The remaining of this paper is organized as follows. In Section 2, we present the principle of a representative slicing algorithm for single-dimension attributes, which serves as the basis for the multi-dimensional slicing presented in Section 3. We present an evaluation of our contribution in Section 4. Related work is surveyed in Section 5. Section 6 concludes the paper.

2 Background

We present in this Section the principles of gossip-based protocols and the algorithm for single-dimension slicing that we extend for supporting multi-dimensional slicing. This algorithm, which we name *sort slicing* is based on a self-organizing gossip-based distributed sort and was initially presented in [13].

Gossip-based distributed systems allow implementing autonomous and self-organizing protocols. An early gossip-based protocol was proposed for the reconciliation of database replicas in the 1980s [6]. Gossip came back into fashion in the 2000s as the complexity of using centralized or rigid management became clear with the increase of systems scales. A typical use of gossip is reliable application-level broadcast [7]. Another class deal with overlay management [15, 16, 20, 21, 24]. Slicing protocols [8, 11, 13, 17] are a subset of it.

In an overlay management gossip-based protocol, each node maintains a small set of links to other nodes. This small set of nodes is called the node's *view*. It is often of fixed size. Each node features an active and a passive thread. The active thread periodically initiates a gossip interaction between the node and another node from the system. The period between two consecutive interactions is defined as a *cycle*. The passive thread is in charge of replying to incoming interaction requests. The interaction typically involves the exchange of information between the two nodes. It results in the two nodes potentially changing their state, or taking some action based on that exchanged information. Gossip-based protocols solely rely on this simple interaction/exchange pattern and do not require further synchronization between nodes. The nature of the information exchanged and the action taken define the nature and goal of a gossip-based protocol.

The partner node for the interaction is either randomly picked from the view, or obtained from a Peer Sampling Service (PSS) [16, 23]. The PSS is

Algorithm 1: Simplified Sorting-Slicing Algorithm on node p [13].

```
1  variables
2      a_p ← ...                                              // local attribute
3      e_p ← rand(0,1)                                        // local position estimation
4      S ← ...                                                // slice schema, array [1..N]: Σ_{i=1}^{N} S[i] = 1
5  every Δ time units do                                      // active thread
6      select partner q randomly from view ∪ PSS.getView()
7      e_p ← q.receive(a_p, e_p)                              // initiate exchange
8  function receive(a_q, e_q)                                 // passive thread
9      if (a_p < a_q ∧ e_p > e_q) ∨ (a_p > a_q ∧ e_p < e_q) then   // order violation?
10         t ← e_p                                            // save current position estimation
11         e_p ← e_q                                          // exchange current position estimation
12         return t
13     else                                                   // no order violation, no exchange
14         return e_q
15 function getSlice()
16     t ← 0, r ← 1
17     while t < e_p do                                       // calculate slice estimation
18         t ← t + S[r], r ← r + 1
19     return r
```

a fundamental service for other gossip-based protocols. It is also itself using a gossip-based implementation. It manages nodes' membership to the system and maintains connectivity. It provides other gossip protocols with a constantly updated stream of random peers drawn from the entire system, forming an overlay that has similar properties to a random graph. It inserts new nodes in the stream of other nodes and ensures that removed nodes do not appear in other nodes' streams after some bounded time. The use of a PSS is mandatory for all gossip-based overlay management protocols to provide convergence and associated self-organizing guarantees [24]: this also applies to gossip-based slicing protocols.

All gossip-based slicing algorithms share the following characteristics. Each node possesses an attribute value a, over a single dimension, that represents one of its characteristics (e.g., the available disk space in Figure 1). The domain of a is unknown and unbounded. It also knows the slicing schema S, a system-wide parameter. Finally, each node's goal is to obtain a position estimation e, that indicates its estimated position in the sorted list of all attribute values, if such a list was materialized. The value of e lies in the bounded domain $[0, 1]$. Knowing S and e, it is straightforward to determine which slice the node belongs to. For instance, with $e = 0.55$ and $S = \{50\%, 30\%, 20\%\}$, node 10 in Figure 1 can determine it belongs to the second slice.

The gossip-based *sort slicing* [13] algorithm determines e based on the following method, detailed in Algorithm 1 for a given node p. The view is composed of

nodes that had numerically close values of e to the current node, at the last time an interaction took place. Interaction partners are selected randomly from nodes in the union of the view and the current stream of nodes exposed by the PSS. The use of the view is only meant to speed-up the convergence, while the use of the PSS is required for that convergence to happen [24]. Each node is bootstrapped with a *random* value for e in $[0, 1]$. The goal of the gossip interactions is to match the ordering of e values on all nodes, to the ordering of a values on all nodes. It implements a distributed swap-sort for this. At each gossip interaction, the two interacting nodes p and q compare the ordering of their values of e and the ordering of their values of a. If the two orders are the same (i.e., if $(a_p < a_q \wedge e_p < e_q) \vee (a_p > a_q \wedge e_p > e_q)$), no action is taken. Otherwise, there is an order violation that needs to be corrected. Nodes proceed to this correction by simply exchanging their position estimations e_p and e_q. The authors of [13] show that this simple method yields an exponential reduction of the disorder metric (the sum of the squared distances of position estimation from their final value) with the number of cycles. This translates in a rapid convergence towards the correct slice attribution at each node.

3 Multi-Dimensional Slicing

We present in this Section our contributions towards autonomous slicing for large-scale distributed systems based on attributes over multiple dimensions. We start by noting that using a utility function does not fit the requirements we have set for multi-dimensional slicing. A utility function maps multiple dimensions onto a single one by applying a linear combination of the values over these dimensions. This requires knowing or estimating the boundaries for each dimension. It also requires assigning weights to metrics (the values over each dimension) that are not comparable. It finally requires making assumptions on the distribution of values over each dimension, as skewed distributions for the value of one attribute would result in the order for other attributes dominating the selection process. We thus dismiss this naive option and consider instead solutions that preserve the multi-dimensional aspect of the workload. First, we report on a negative result on our initial use of space-filling curves. Then, we present our approach using domination relations between multi-dimensional nodes' attributes, the resulting partial order and order violation resolutions, and the resulting multi-dimensional slicing.

Using space filling curves? Our first attempt to provide multi-dimensional slicing was to map attributes of multiple dimensions onto a single dimension using space filling curves (SFC). A SFC is a curve that covers the entire unit square. It can be generalized to a d-dimensional space. Each d-dimensional point in the space is associated with a single point on the curve. The distance on the curve of the representations of two d-dimensional points can be used to estimate the distance between the two original d-dimensional points in the space. SFC are typically defined by a recursive definition, as a special case of fractal construction.

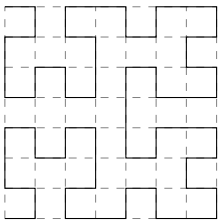

Fig. 3. 2-D Hilbert SFC.

There exists many SFCs but we chose to use the Hilbert SFC (illustrated for 2 dimensions on Figure 3). This curve has the property that it tends to preserve ordering for a large number, but not all, possible couple of points mapped on the curve with respect to the ordering of their corresponding vector to the origin in the space. This initial approach proved unsatisfactory for the following two reasons. First, a major issue with SFC is that they require knowing in advance –or estimating– the boundaries of the domain space. This condition is not met by typical workloads for resource provisioning in large-scale systems. Second, while the use of a SFC allows forming a total order between multi-dimensional points, the ordering inconsistencies (e.g., between [2:1], [3:3] and [2,2] on Figure 3) prevent the algorithm from converging towards stable position estimations. As a result, we discarded the SFC approach and looked into a solution that intrinsically does not feature these two limitations, and does not require mapping multiple dimensions into a single one.

3.1 Domination Relations

Our approach for multi-dimensional slicing is based on the intuitive notion of quality of compromises that we presented in our problem definition. It does not require knowing the range of definition of any of the dimensions. It is based on a domination relation $\lhd$. We say that a node n_a dominates another node n_b when n_a is strictly a better compromise than n_b according to the value of their attributes over all dimensions. The definition of $\lhd$ is as follows, where n_a^i denotes the value of the attribute of n_a for dimension i and where d is the number of dimensions: $n_a \lhd n_b$ if $\forall\ i \mid 1 \leq i \leq d, n_a^i \leq n_b^i$. This relation is transitive: $n_a \lhd n_b \wedge n_b \lhd n_c \Rightarrow n_a \lhd n_c$ and its inverse is $\rhd$: $n_a \lhd n_b \Leftrightarrow n_b \rhd n_a$.

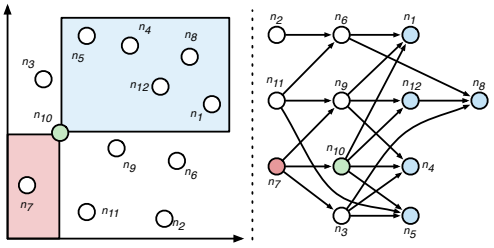

Fig. 4. Domination relations and partial order.

Figure 4 (left) gives an example of domination relations between a node n_{10} and a set of other nodes, over a 2-dimensional space. Relations that can be determined based on transitivity are not shown. In particular, we have $n_{10} \lhd \{n_1, n_4, n_5, n_8, n_{12}\}$ and $n_{10} \rhd \{n_7\}$. The $\lhd$ relations define a partial order $\mathcal{O}_\lhd$ between nodes. The partial order for nodes $n_1, \ldots, n_{12}$ is shown on the right side of Figure 4: we note that some pairs of nodes have no defined ordering according to $\lhd$ and $\rhd$. For instance, n_{11} and n_7 have no established $\lhd$ or $\rhd$ relation.

We note that our notion of domination between nodes is similar to the notion of domination between data points that is used in the context of databases,

Algorithm 2: Multi-dimensional domination-based slicing algorithm on node p (*getSlice()* is unchanged from Algorithm 1).

```
1  variables                                          // in addition to the ones in Algorithm 1
2      a_p = {a_p^1, ..., a_p^d} ← ...                // local multi-dimensional attribute
3      ◁-view ← ∅                                     // view, close dominated nodes
4      ▷-view ← ∅                                     // view, close dominating nodes
5  every Δ time units do                              // active thread
6      select partner q randomly from ◁-view ∪ ▷-view ∪ PSS.getView()
7      e_p ← q.receive(a_p, e_p, p)                   // initiate exchange
8  function receive(a_q, e_q, q)                      // passive thread
9      if a_p ◁ a_q then
10         ◁-view ← ◁-view ∪ q                        // maintain close dominated nodes view
11         foreach n ∈ ◁-view do
12             if ∃ n' ∈ ◁-view | n ≠ n' ∧ a_n' ◁ a_n then
13                 ◁-view ← ◁-view \ n
14         if e_q < e_p then return exchange(e_q)     // order violation?
15     else if a_p ▷ a_q then
16         ▷-view ← ▷-view ∪ q                        // maintain close dominating nodes view
17         foreach n ∈ ▷-view do
18             if ∃ n' ∈ ▷-view | n ≠ n' ∧ a_n' ▷ a_n then
19                 ▷-view ← ▷-view \ n
20         if e_q > e_p then return exchange(e_q)     // order violation?
21     return(e_q)                                    // no order violation, no exchange
22 function exchange(e_q)
23     t ← e_p, e_p ← e_q, return t
```

and in particular for the definition of SkyLine queries [12]. A SkyLine query allows collecting the best-compromise points from a structured multi-dimensional database: the SkyLine points are the points that are not dominated by any other point, under the same definition as above. The SkyLine of Figure 4 would be $\{n_7, n_{11}, n_2\}$. The SkyLine operator has been extended towards the SkyBand operator [22]. A k-SkyBand is a set of points that are not dominated by more than k points. The 2-SkyBand of Figure 4 would be $\{n_7, n_{11}, n_2, n_6, n_9, n_{10}, n_3\}$. The slicing problem is however different than the k-SkyBand in several ways. First, there is no omniscient view of the data set as in a database. Nodes only know a small amount of other nodes in their view and have no global knowledge of the boundaries of the space, the number of nodes, or the distribution of values for each dimension. Second, the slices are not defined in terms of the number of dominating nodes but in terms of a size relative to the totality of the system. We detail below how we modify the uni-dimensional slicing from Algorithm 1 to support multi-dimensional slicing based on domination relations.

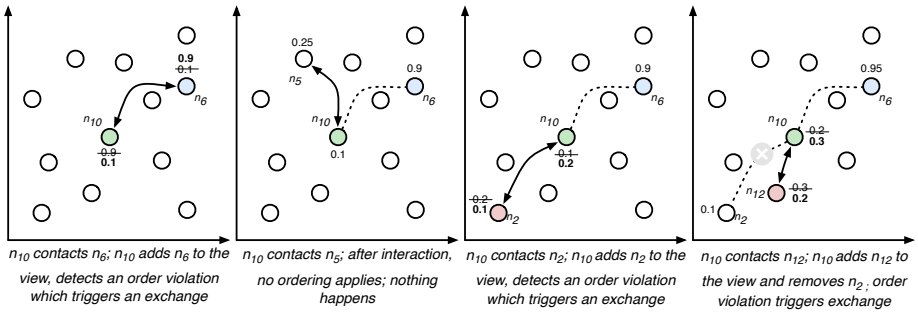

Fig. 5. Evolution of the view of a node according to Algorithm 2.

3.2 Multi-dimensional Slicing from Domination-based Ordering

The multi-dimensional slicing algorithm is presented by Algorithm 2. It extends the previous Algorithm 1 for slicing over one dimension. An illustration of the algorithm is provided by Figure 5. Description of the illustrated algorithm steps is directly provided below the figure itself. Similarly as before, each node p has access to a PSS and maintains a view. This view of p is split in two parts: the $\triangleleft$-view and the $\triangleright$-view. The former holds the peers that are directly dominated by p and by no other node in $\triangleleft$-view. The latter has the same semantics for dominating peers under the $\triangleright$ relation. Each time a gossip exchange takes place, if the domination relation $\triangleleft$ holds between p and the exchange partner q, q is first added to $\triangleleft$-view (line 10). The $\triangleleft$-view is then pruned out of nodes that are already dominated by another one from the set (lines 11 to 13). The same principle is applied to dominating nodes and the $\triangleright$-view (lines 15 to 19). When an order violation is detected (line 14 or line 20), the position estimation e_p and e_q are exchanged, in the same way as for Algorithm 1. The slice determination itself is strictly identical as for Algorithm 1.

Algorithm 2 can be enhanced in a number of ways. First, instead of keeping the directly dominated nodes and directly dominating nodes in $\triangleleft$-view and $\triangleright$-view, we can keep a set of nodes to ensure a faster resolution of "local" order violations ("global" order violations between nodes are expected to be resolved more frequently due to the interactions from the PSS, but close links help in the final steps of convergence as demonstrated in [24]). To this end, node p might keep a set of peers that is not actually the Skyline of the dominated (resp. dominating) nodes but an overset of it, nodes are removed from the $\triangleleft$-view and $\triangleright$-view only when it goes over a target view size (by modifying the conditions on lines 12 and 18). Second, the interaction depicted is single-way: while both p and q participate in the exchange, only p takes advantage of it to fill its views as part of its passive thread (this is known as a push-only operation). It is straightforward for q to operate in the same manner while on its active thread (push-pull). We use these two optimizations in our implementation.

We note that several of the interactions with random nodes chosen using the PSS will lead to no action as the two values will not be comparable in the partial order $\mathcal{O}_{\triangleleft}$.[4] These random interactions are necessary however when the attributes value change over time, as expected in a dynamic provisioning scenario (e.g., the uptime, available disk space, etc. evolve over time). Unlike with uni-dimensional slicing, no total order on the nodes exists. This means that there are a multitude of possible total ordering of the position estimates e_p that are compatible with the partial ordering $\mathcal{O}_{\triangleleft}$. As we point out in our evaluation, this also means that the scheme is unsurprisingly subject to the *curse of dimensionality* [3]. When using over 8 or 10 dimensions, the partial order tends to become a very sparse graph as most nodes are not comparable against $\mathcal{O}_{\triangleleft}$ anymore. We believe however that the problem of dynamic node provisioning in large-scale systems does not require more than this limit number of dimensions to fulfill its intended role.

4 Evaluation

We now present the evaluation of our protocol. Our implementation uses PeerSim [1], a Java-based cycle-oriented simulation framework that is well adapted to the study of gossip-based protocols. We simulate a network of 10,000 nodes for all the presented experiments. Each node uses an attribute over 2 to 15 dimensions (when not specified, we present results for 2 dimensions). Values for each dimension are random float numbers. Each node has access to an independent PSS. We use an instantiation of the Cyclon [23] PSS in the framework of [16]. Each node maintains a $\triangleleft$-view and a $\triangleright$-view of up to 10 entries each.

The reminder of this section is organized as follows. We start by evaluating the protocol correctness. Then, we evaluate the convergence speed with different view configurations. Finally we study the scalability in terms of number of dimensions.

Correctness. We evaluate the correctness of the protocol by its capacity to reduce the number of *order violations* as defined in Section 3.1. An order violation between two nodes p and q with position estimates e_p and e_q occurs when $((p \triangleleft q \wedge e_p > e_q) \vee (p \triangleright q \wedge e_p < e_q))$. We count the number of such violations per node when compared in an omniscient way against all other nodes in the system. Note that we cannot use a disorder metric as in [13] as the partial order between nodes is not based on an Euclidean distance but on domination relationships. Figure 6 depicts the evolution of the average and maximum number of order violations for nodes in the system, as nodes engage in cycles of gossip exchanges (during one cycle, each node initiates one gossip exchange with one partner). As expected from previous work on gossip-based overlay construction protocols [8, 11, 15–17, 20, 21, 24] and similarly to the uni-dimensional slicing in [13], the reduction in the number of violations is exponential. After only 10

[4] The choice of the partner for the gossip interaction could be based on semantic information piggybacked on the PSS messages. This requires however one more round of interaction to check that the actual known attribute values are up to date (as detailed in [13]). We did not implement this optimization in our prototype.

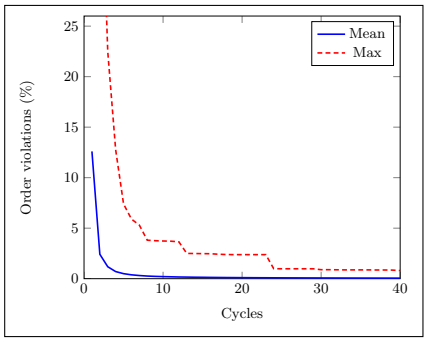

Fig. 6. Convergence speed.

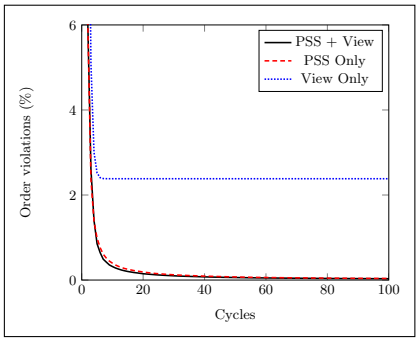

Fig. 7. Impact of the PSS.

cycles, the system converges to less than 0.05% violations per node on average and 3.7% as worst case. This indicates that the node provisioning quality will highly resemble the "ideal" slicing for the given attributes values set. We note that the max converges slowly to 0: this is a result of remaining violations being slowly resolved by using random PSS links in a network where a majority of nodes have converged views. This behavior was pointed out in [24]. It does not pose a significant problem in practice as it has little influence on the slicing decision itself, which is already converged as we see in our last experiment.

Influence of the gossip interaction partner selection. Figure 7 illustrates the influence of the selection of the gossip interaction partner (line 6 in Algorithm 2) on the convergence speed and correctness. We use the same metric as before and present the mean number of violations for the three following cases. The gossip exchanges are initiated by randomly picking a node from the view only (configured by a static bootstrapping using random peers), using nodes obtained from the PSS only, or using a combination of both. As expected, we observe that the convergence *requires* the use of the PSS. Using only the views bootstrapped with initial random nodes but not constantly updated with new random nodes, the system ends up in a state where nodes with order violations lie in disconnected islands and never get to resolve these violations anymore, preventing the whole system from converging. We also observe that the use of the view for choosing the gossip partner has a slight, but almost negligible impact on the convergence speed. These two observations are in line with the observations made for other gossip-based overlay construction protocols [15, 20, 24].

Scalability in number of dimensions. Our next series of experiments evaluate the impact of the number of dimensions used for nodes' attributes on the stability and observed convergence speed. Figure 8 reports the evolution of the number of order violations when using 2, 4, 6 and 15 dimensions. More dimensions mean less possible comparisons between nodes against $\lhd$. The convergence speed remains good with up to 6 dimensions. As expected, when using a large number of dimensions (in general, starting from 8), an embarrassing majority of couples

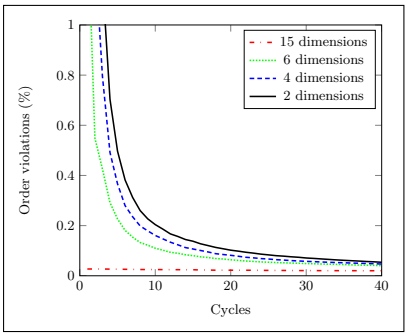

Fig. 8. Convergence vs dimension.

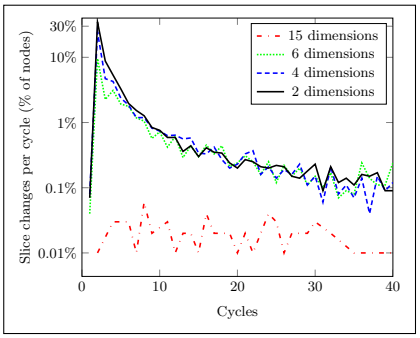

Fig. 9. Slice allocation stability.

of nodes are not comparable with $\lhd$. This means that the slicing operation using domination-based order is no longer possible, unless one restricts the operation to a subset of representative dimensions. We believe 6 dimensions is enough characteristics to consider for dynamic node provisioning in large-scale systems.

Finally, we evaluate the impact of the number of dimensions on the stability of the system, in terms of the steadiness of the slice allocation for each node. Figure 9 presents the number of slice changes per cycle (note the logarithmic scale) as the protocol converges. The cost of a slice depends on the supported application. In general, a good practice is to export the slice allocation only when it has been steady for a configurable number of cycles. With 15 dimensions, nodes are basically stuck with the slice that corresponds to their initial random value for the position estimation p, as there are almost no order violations to resolve. Along with the previous observation, this exemplifies the expected effect of the *curse of dimensionality* [3] on multi-dimensional slicing. With 1 to 6 dimensions though, we see that the system quickly converges to stable slice allocations. The protocol starts its execution after the 2nd cycle, where 40% of the nodes decide to change their slide. After about 10 cycles, only 100 slice changes per cycle happen. This concerns a mere 1% of the network. After 30 cycles, there are only 10 such changes (0.1%). In a practical deployment, one can freeze the slice allocation around these times depending on the required precision.

5 Related Work

Besides the *sort slicing* protocol that we described in Section 2, the following slicing protocols were proposed. Ranking [8] is not based on the exchange of position estimates but on counting mechanisms that estimate how many nodes have lower, resp. higher values for the single-dimension attribute. Sliver [11] improves this protocol by introducing a technique to improve slicing accuracy avoiding mistakenly considering duplicate information. Slead [17] is an extension of this idea that make use of Bloom filters to reduce the high memory consumption of these protocols and proposes a hysteresis mechanisms for the slice determination

to gain more stability. All these protocols are based on the gossip-based paradigm, for which we covered related work in Section 2.

We review below alternative proposals to slicing for large-scale system resource allocation. Minerva [2] contributes tools to efficiently assign host machines to support large-scale storage systems. The assignment of the hosts is centrally executed by an *allocator* node. Our approach relies on autonomous decisions taken by each participants of the protocol, without any central coordinator. The autonomous resource selection presented in [5] organizes nodes in a peer-to-peer system over a multi-dimensional space, upon which lookup queries are issued to select nodes matching the specified requirements. Similarly to ours, this protocol has built-in support for multi-dimensional attributes. This system solves a related but slightly different problem than the one addressed in this paper, that is the selection of a fixed number of nodes that satisfy the specified queries, and does not rely on self-organizing techniques but explicit query/response mechanisms.

In [10], a notion of resource dominance is also used. It differs however from the domination between nodes we use in our work. Domination in [10] relates to the prevalence of the usage of one type of resource (CPU, memory, ...) over the other types of resources on one node. This information is used by a centralized resource allocator in order to achieve resource allocation fairness for a given workload.

6 Conclusion

Distributed slicing is a fundamental primitive for resource allocation in large-scale distributed systems. Existing distributed slicing protocols did not offer support to perform this allocation considering nodes with multidimensional attributes. In practical settings, it is often desired to choose a best compromise between various characteristics (i.e. CPU, volatile memory, uptime, etc.). We contribute a gossip-based distributed slicing protocol that fills this need. The proposed protocol exploits domination relations between nodes to define the membership of a node to a given slice. Our evaluation confirms the contributed solution as a sound approach to the problem. We support experimental reproducibility: source code and datasets are released under open-source and available at: `https://github.com/mpasquet/multidimslicing_dais2014`.

Acknowledgments. This work received support from the Portuguese Foundation for Science and Technology under grant SFRH/BD/71476/2010.

References

1. `http://peersim.sourceforge.net/`.
2. Alvarez, G. A., Borowsky, E., Go, S., Romer, T. H., Becker-Szendy, R., Golding, R., Merchant, A., Spasojevic, M., Veitch, A., and Wilkes, J. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems (TOCS) 19*, 4 (2001), 483–518.

3. Bellman, R. Curse of dimensionality. *Adaptive control processes: a guided tour. Princeton, NJ* (1961).
4. Bhagwan, R., Savage, S., and Voelker, G. M. Understanding availability. In *IPTPS 2003*.
5. Costa, P., Napper, J., Pierre, G., and Steen, M. v. Autonomous resource selection for decentralized utility computing. In *ICDCS 2009*.
6. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. Epidemic algorithms for replicated database maintenance. In *PODC 1987*.
7. Eugster, P. T., Guerraoui, R., Handurukande, S. B., Kouznetsov, P., and Kermarrec, A.-M. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst. 21*, 4 (Nov. 2003), 341–374.
8. Fernandez, A., Gramoli, V., Jimenez, E., Kermarrec, A.-M., and Raynal, M. Distributed slicing in dynamic systems. In *ICDCS 2007*.
9. Garbacki, P., Iosup, A., Epema, D., and van Steen, M. 2Fast: Collaborative Downloads in P2P Networks. In *IEEE P2P 2006*.
10. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., and Stoica, I. Dominant resource fairness: Fair allocation of multiple resource types. In *USENIX NSDI 2011*.
11. Gramoli, V., Vigfusson, Y., Birman, K., Kermarrec, A.-M., and van Renesse, R. Slicing distributed systems. *IEEE Trans. Comput. 58*, 11 (Nov. 2009).
12. Hose, K., and Vlachou, A. A survey of skyline processing in highly distributed environments. *The VLDB Journal 21*, 3 (2012), 359–384.
13. Jelasity, M., and Kermarrec, A.-M. Ordered slicing of very large-scale overlay networks. In *IEEE P2P 2006*.
14. Jelasity, M., Montresor, A., and Babaoglu, O. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems 23*, 3 (2005).
15. Jelasity, M., Montresor, A., and Babaoglu, O. T-man: Gossip-based fast overlay topology construction. *Computer Networks 53*, 13 (aug 2009), 2321–2339.
16. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., and Van Steen, M. Gossip-based peer sampling. *ACM Transactions on Computer Systems (TOCS) 25*, 3 (2007), 8.
17. Maia, F., Matos, M., Rivière, E., and Oliveira, R. Slead: low-memory, steady distributed systems slicing. In *DAIS 2012*.
18. Maia, F., Matos, M., Vilaça, R., Pereira, J., Oliveira, R., and Rivière, E. Dataflasks: an epidemic dependable key-value substrate. In *DCDV 2013*.
19. Matos, M., Schiavoni, V., Felber, P., Oliveira, R., and Rivière, E. Lightweight, efficient, robust epidemic dissemination. *Journal of Parallel and Distributed Computing 73*, 7 (2013), 987–999.
20. Montresor, A., Jelasity, M., and Babaoglu, O. Chord on demand. In *IEEE P2P 2005*.
21. Montresor, A., Jelasity, M., and Babaoglu, O. Decentralized ranking in large-scale overlay networks. In *SASOW 2008*.
22. Vlachou, A., Doulkeridis, C., Norvag, K., and Vazirgiannis, M. Skyline-based peer-to-peer top-k query processing. In *ICDE 2008*.
23. Voulgaris, S., Gavidia, D., and Steen, M. V. Cyclon: Inexpensive membership management for unstructured P2P overlays. *Journal of Network and Systems Management* (2005).
24. Voulgaris, S., and van Steen, M. VICINITY: A pinch of randomness brings out the structure. In *Middleware 2013*.