HAL open science

Distributed Slicing in Dynamic Systems

Antonio Fernández, Vincent Gramoli, Ernesto Jimenez, Anne-Marie Kermarrec, Michel Raynal

To cite this version:

Antonio Fernández, Vincent Gramoli, Ernesto Jimenez, Anne-Marie Kermarrec, Michel Raynal. Distributed Slicing in Dynamic Systems. [Research Report] FGJKR06, 2006, pp.27. inria-00118675v1

HAL Id: inria-00118675

https://inria.hal.science/inria-00118675v1

Submitted on 6 Dec 2006 (v1), last revised 6 Dec 2006 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

inria-00118675, version 1 - 6 Dec 2006

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Distributed Slicing in Dynamic Systems

Antonio Fernández — Vincent Gramoli — Ernesto Jiménez —

Anne-Marie Kermarrec — Michel Raynal

N° ????

Décembre 2006

Thème COM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--????--FR+ENG

Distributed Slicing in Dynamic Systems

Antonio Fernández[*], **Vincent Gramoli**[†], **Ernesto Jiménez**[‡], **Anne-Marie Kermarrec**[†], **Michel Raynal**[†]

Thème COM — Systèmes communicants
Projet Asap

Rapport de recherche n° ???? — Décembre 2006 — 26 pages

Abstract: Peer to peer (P2P) systems are moving from application specific architectures to a generic service oriented design philosophy. This raises interesting problems in connection with providing useful P2P middleware services that are capable of dealing with resource assignment and management in a large-scale, heterogeneous and unreliable environment. One such service, the slicing service, has been proposed to allow for an automatic partitioning of P2P networks into groups (slices) that represent a controllable amount of some resource and that are also relatively homogeneous with respect to that resource, in the face of churn and other failures. In this report we propose two algorithms to solve the distributed slicing problem. The first algorithm improves upon an existing algorithm that is based on gossip-based sorting of a set of uniform random numbers. We speed up convergence via a heuristic for gossip peer selection. The second algorithm is based on a different approach: statistical approximation of the rank of nodes in the ordering. The scalability, efficiency and resilience to dynamics of both algorithms relies on their gossip-based models. We present theoretical and experimental results to prove the viability of these algorithms.

Key-words: Slicing, Gossip, Slice, Churn, Peer-to-Peer, Aggregation, Large Scale, Resource Allocation.

Contact author: Vincent Gramoli `vgramoli@irisa.fr`

[*] Universidad Rey Juan Carlos, 28933 Móstoles, Spain. anto@gsyc.escet.urjc.es
[†] IRISA, INRIA Université Rennes 1 (ASAP Research Group) 35042 Rennes, France. {vgramoli,akermarr,raynal}@irisa.fr
[‡] Universidad Politécnica de Madrid, 28031 Madrid, Spain. ernes@eui.upm.es

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Morcellement distribué dans les systèmes dynamiques

Résumé : Un service de *morcellement* d'un réseau pair-à-pair permet de partitionner les nœuds du système en plusieurs groupes appelés *morceaux*. Ce rapport présente deux algorithmes pour résoudre le problème du morcellement réparti. Le premier algorithme améliore un algorithme existant en accérant son temps de convergence. Le second algorithme utilise une approche différente d'appoximation statistique. Des résultats théoriques et experimentaux montrent la viabilité de nos algorithmes.

Mots-clés : Morcellement, Bavardage, Morceau, Va-et-vient, Pair-à-pair, Aggrégation, Grande échelle, Allocation de ressources.

Distributed Slicing in Dynamic Systems

December 6, 2006

1 Introduction

1.1 Context and Motivations

The peer to peer (P2P) communication paradigm has now become the prevalent model to build large-scale distributed applications, able to cope with both scalability and system dynamics. This is now a mature technology: peer to peer systems are slowly moving from application-specific architectures to a generic-service oriented design philosophy. More specifically, peer to peer protocols hold the promise to integrate into platforms on top of which several applications with various requirements may cohabit. This leads to the interesting issue of resource assignment or how to allocate a set of nodes for a given application. Examples of targeted platforms for such a service are telecommunication platforms, where some set of peers may be automatically assigned to a specific task depending on their capabilities, testbed platform such as Planetlab [2], or desktop-grid-like applications [1].

In this context, the ordered slicing service has been recently proposed as a building block to allocate resources, i.e a set of nodes sharing some characteristics with respect to a given metric or attribute, in a large-scale peer to peer system. This service acknowledges the fact that peers potentially offer heterogeneous capabilities as revealed by many recent works describing heavy-tailed distributions of storage space, bandwidth, and uptime of peers [16, 3, 17]. The slicing service [10] enables peers in a large-scale unstructured network to self-organize into a partitioning, where partitions (slices) are connected overlay networks that represent a given percentage of some resource. Such slices can be allocated to specific applications later on. The slicing is ordered in the sense that peers get ranked according to their capabilities expressed by an attribute value.

Large scale dynamic distributed systems consist of many participants that can join and leave at will. Identifying peers in such systems that have a similar level of power or capability (for instance, in terms of bandwidth, processing power, storage space, or uptime) in a completely decentralized manner is a difficult task. It is even harder to maintain this information in the presence of churn. Due

*Universidad Rey Juan Carlos, 28933 Móstoles, Spain. anto@gsyc.escet.urjc.es
†IRISA, INRIA Université Rennes 1 (ASAP Research Group) 35042 Rennes, France. {vgramoli,akermarr,raynal}@irisa.fr
‡Universidad Politécnica de Madrid, 28031 Madrid, Spain. ernes@eui.upm.es

to the intrinsic dynamics of contemporary peer to peer systems it is impossible to obtain accurate information about the capabilities (or even the identity) of the system participants. Consequently, no node is able to maintain accurate information about all the nodes. This disqualifies centralized approaches.

Taking this into account, we can summarize the ordered slicing problem we tackle in this report: we need to rank nodes depending on their capability, slice the network depending on these capabilities and, most importantly, readapting the slices continuously to cope with system dynamism.

Building upon the work on ordered distributed slicing proposed in [10], here we focus on the issue of *accurate* slicing. That is, we focus on improving the quality and stability of the slices, both aspects being crucial for potential applications.

1.2 Contributions

The report presents two gossip-based solutions to slice the nodes according to their capability (reflected by an attribute value) in a distributed manner with high probability. The first contribution of the report builds upon the distributed slicing algorithm proposed in [10] that we call the JK algorithm in the sequel of this report. The second algorithm is a different approach based on rank approximation through statistical sampling.

In JK, each node i maintains a random number r_i, picked up uniformly at random (between 0 and 1), and an attribute value a_i, expressing its capability according to a given metric. Each peer periodically gossips with another peer j, randomly chosen among the peers it knows about. If the order between r_j and r_i is different than the order between a_j and a_i, random values are swapped between nodes. The algorithm ensures that eventually the order on the random values matches the order of the attribute ones. The quality of the ranking can then be measured by using a global disorder measure expressing the difference between the exact rank and the actual rank of each peer along the attribute value.

The first contribution of this report is to propose a local disorder measure so that a peer chooses the neighbor to communicate with in order to maximize the chance of decreasing the global disorder measure. The interest of this approach is to speed the convergence up. We provide the analysis and experimental results of this improvement.

Once peers are ordered along the attribute values, the slicing in JK takes place as follows. Random values are used to calculate which slice a node belongs to. For example, a slice containing 20% of the best nodes according to a given attribute, will be composed of the nodes that end up holding random values greater than 0.8. The accuracy of the slicing (independent from the accuracy of the ranking) fully depends on the uniformity of the random value spread between 0 and 1 and the fact that the proportion of random values between 0.8 and 1 is approximately (but usually not exactly) 20% of the nodes. Another contribution of this report is to precisely characterize the potential inaccuracy resulting from this effect.

This observation means that the problem of ordering nodes based on uniform random values is not fully sufficient for determining slices. This motivates us to find an alternative approach to this algorithm and JK in order to determine more precisely the slice each node belongs to.

Another motivation for an alternative approach is related to churn and dynamism. It may well happen that the churn is actually correlated to the attribute value. For example, if the peers are sorted according to their connectivity potential, a portion of the attribute space (and therefore the random value space) might be suddenly affected. New nodes will then pick up new random values and eventually the distribution of random values will be skewed towards high values.

The second contribution is an alternative algorithm solving these issues by approximating the rank of the nodes in the ordering locally, without the application of random values. The basic idea is that each node periodically estimates its rank along the attribute axis depending of the attributes it has seen so far. This algorithm is robust and lightweight due to its gossip-based communication pattern: each node communicates periodically with a restricted dynamic neighborhood that guarantees connectivity and provides a continuous stream of new samples. Based on continuously aggregated information, the node can determine the slice it belongs to with a decreasing error margin. We show that this algorithm provides accurate estimation at the price of a slower convergence.

1.3 Outline

The rest of the report is organized as follows: Section 2 surveys some related work. The system model is presented in Section 3. The first contribution of an improved ordered slicing algorithm based on random values is presented in Section 4 and the second algorithm based on dynamic ranking in Section 5. Section 6 concludes the report.

2 Related Work

Most of the solutions proposed so far for ordering nodes come from the context of databases, where parallelizing query executions is used to improve efficiency. A large majority of the solutions in this area rely on centralized gathering or all-to-all exchange, which makes them unsuitable for large-scale networks. For instance, the *external sorting problem* [5] consists in providing a distributed sorting algorithm where the memory space of each processor does not necessarily depend on the input. This algorithm must output a sorted sequence of values distributed among processors. The solution proposed in [5] needs a global merge of the whole information, and thus it implies a centralization of information. Similarly, the *percentile finding* problem [8], which aims at dividing a set of values into equally sized sets, requires a logarithmic number of all-to-all message exchanges.

Other related problems are the selection problem and the ϕ-quantile search. The selection problem [6, 4] aims at determining the ith smallest element with as few comparisons as possible. The *ϕ-quantile* search (with $\phi \in (0, 1]$) is the problem to find among n elements the $(\phi n)^{th}$ element. Even though these problems look similar to our problem, they aim at finding a specific node among all, while the distributed slicing problem aims at solving a global problem where each node maintains a piece of information. Additionally, solutions to the quantile search problem like the one presented in [13] use an approximation of the system size. The same holds for the algorithm in [15], which uses similar ideas to determine the distribution of a utility in order to isolate peers with high capability—i.e., super-peers.

As far as we know, the distributed slicing problem was studied in a P2P system for the first time in [10]. In this report, a node with the kth smallest attribute value, among those in a system of size n, tries to estimate its normalized index k/n. The *JK algorithm* proposed in [10] works as follows. Initially, each node draws independently and uniformly a random value in the interval $(0, 1]$ which serves as its first estimate of its normalized index. Then, the nodes use a variant of Newscast [12] to gossip among each other to exchange random values when they find that the relative order of their random values and that of their attribute values do not match. This algorithm is robust in face of frequent dynamics and guarantees a fast convergence to the same sequence of peers with respect to the random and the attribute values. At every point in time the current random value of a node serves to estimate the slice to which it belongs (its slice).

3 Model

3.1 System model

We consider a system Σ containing a set of n uniquely identified nodes. (Value n may vary over time, dynamics is explained below). The set of identifiers is denoted by I. Each node can leave and new nodes can join the system at any time, thus the number of nodes is a function of time. Nodes may also crash. In this report, we do not differentiate between a crash and a voluntary node departure.

Each node i maintains an attribute value a_i, reflecting the node capability according to a specific metric. These attribute values over the network might have an arbitrary skewed distribution. Initially, a node has no global information neither about the structure or size of the system nor about the attribute values of the other nodes.

We can define a total ordering over the nodes based on their attribute value, with the node identifier used to break ties. Formally, we let i precede j if and only if $a_i < a_j$, or $a_i = a_j$ and $i < j$. We refer to this totally ordered sequence as the *attribute-based sequence*, denoted by $A.sequence$. The attribute-based rank of a node i, denoted by $\alpha_i \in \{1, ..., n\}$, is defined as the index of a_i in $A.sequence$. For instance, let us consider three nodes: 1, 2, and 3, with three different attribute values $a_1 = 50$, $a_2 = 120$, and $a_3 = 25$. In this case, the attribute-based rank of node 1 would be $\alpha_1 = 2$. In the rest of the report, we assume that nodes are sorted according to a single attribute and that each node belongs to a unique slice. The sorting along several attributes is out of the scope of this report.

3.2 Distributed Slicing Problem

Let $\mathcal{S}_{l,u}$ denote the *slice* containing every node i whose normalized rank, namely $\frac{\alpha_i}{n}$, satisfies $l < \frac{\alpha_i}{n} \leq u$ where $l \in [0, 1)$ is the slice lower boundary and $u \in (0, 1]$ is the slice upper boundary so that all slices represent adjacent intervals $(l_1, u_1], (l_2, u_2]$... Let us assume that we partition the interval $(0, 1]$ using a set of slices, and this partitioning is known by all nodes. The distributed slicing problem requires each node to determine the slice it currently belongs to. Note that the problem stated this way is similar to the ordering problem, where each node has to determine its own index

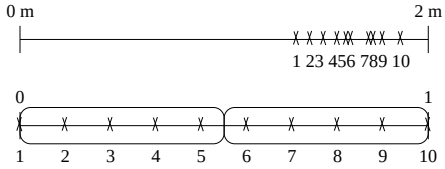

Figure 1: Slicing of a population based on a height attribute.

in $A.sequence$. However, the reference to slices introduces special requirements related to stability and fault tolerance, besides, it allows for future generalizations when one considers different types of categorizations.

Figure 1 illustrates an example of a population of 10 persons, to be sorted against their height. A partition of this population could be defined by two slices of the same size: the group of short persons, and the group of tall persons. This is clearly an example where the distribution of attribute values is skewed towards 2 meters. The rank of each person in the population and the two slices are represented on the bottom axis. Each person is represented as a small cross on these axes.[1] Each slice is represented as an oval. The slice $S_1 = \mathcal{S}_{0,\frac{1}{2}}$ contains the five shortest persons and the slice $S_2 = \mathcal{S}_{\frac{1}{2},1}$ contains the five tallest persons.

Observe that another way of partitioning the population could be to define the group of short persons as that containing all the persons shorter than a predefined measure (e.g., $1.65m$) and the group of tall persons as that containing the persons taller than this measure. However, this way of partitioning would most certainly lead to an unbalanced distribution of persons, in which, for instance a group might be empty (while a slice is almost surely non-empty). Since the distribution of attribute values is unknown and hard to predict, defining relevant groups is a difficult task. For example, if the distribution of the human heights were unknown, then the persons taller than $1m$ could be considered as tall and the persons shorter than $1m$ could be considered as short. Conversely, slices partition the population into subsets representing a predefined portion of this population. Therefore, in the rest of the report, we consider slices as defined as a proportion of the network.

3.3 Facing Churn

Node churn, that is, the continuous arrival and departure of nodes is an intrinsic characteristic of peer to peer systems and may significantly impact the outcome, and more specifically the accuracy of the slicing algorithm. The easier case is when the distribution of the attribute values of the departing and arriving nodes are identical. In this case, in principle, the arriving nodes must find their slices, but the nodes that stay in the system are mostly able to keep their slice assignment. Even in this case however, nodes that are close to the border of a slice may expect frequent changes in their slice due to the variance of the attribute values, which is non-zero for any non-constant distribution. If the arriving and departing nodes have different attribute distributions, so that the distribution in the actual network of live nodes keeps changing, then this effect is amplified. However, we believe that

[1] Note that the shortest (resp. largest) rank is represented by a cross at the extreme left (resp. right) of the bottom axis.

this is a realistic assumption to consider that the churn may be correlated to some specific values (for example if the considered attribute is uptime or connectivity).

4 Dynamic Ordering by Exchange of Random Values

This section proposes an algorithm for the distributed slicing problem improving upon the original JK algorithm [10], by considering a local measure of the global disorder function. In this section we present the algorithm along with the corresponding analysis and simulation results.

4.1 On Using Random Numbers to Sort Nodes

This Section presents the algorithm built upon JK. We refer to this algorithm as *mod-JK* (standing for modified JK). In JK, each node i generates a number $r_i \in (0, 1]$ independently and uniformly at random. The key idea is to sort these random numbers with respect to the attribute values by swapping these random numbers between nodes, so that if $a_i < a_j$ then $r_i < r_j$. Eventually, the attribute values (that are fixed) and the random values (that are exchanged) should be sorted in the same order. That is, each node would like to obtain the xth largest random number if it owns the xth largest attribute value. Let $R.sequence$ denote the *random sequence* obtained by ordering all nodes according to their random number. Let $\rho_i(t)$ denote the index of node i in $R.sequence$ at time t. When not required, the time parameter is omitted.

To illustrate the above ideas, consider that nodes 1, 2, and 3 from the previous example have three distinct random values: $r_1 = 0.85$, $r_2 = 0.1$, and $r_3 = 0.35$. In this case, the index ρ_1 of node 1 would be 3. Since the attribute values are $a_i = 50$, $a_2 = 120$, and $a_3 = 25$, the algorithm must achieve the following final assignment of random numbers: $r_1 = 0.35$, $r_2 = 0.85$, and $r_3 = 0.1$.

Once sorted, the random values are used to determine the portion of the network a peer belongs to.

4.2 Definitions

View. Every node i keeps track of some neighbors and their age. The *age* of neighbor j is a timestamp, t_j, set to 0 when j becomes a neighbor of i. Thus, node i maintains an array containing the id, the age, the attribute value, and the random value of its neighbors. This array, denoted $\mathcal{N}_i$, is called the *view* of node i. The views of all nodes have the same size, denoted by c.

Misplacement. A node participates in the algorithm by exchanging its rank with a misplaced neighbor in its view. Neighbor j is misplaced if and only if

- $a_i > a_j$ and $r_i < r_j$, or
- $a_i < a_j$ and $r_i > r_j$.

We can characterize these two cases by the predicate $(a_j - a_i)(r_j - r_i) < 0$.

Global Disorder Measure. In [10], a measure of the relative disorder of sequence $R.sequence$ with respect to sequence $A.sequence$ was introduced, called the *global disorder measure (GDM)* and defined, for any time t, as

$$GDM(t) = \frac{1}{n}\sum_i (\alpha_i - \rho(t)_i)^2.$$

The minimal value of GDM is 0, which is obtained when $\rho(t)_i = \alpha_i$ for all nodes i. In this case the attribute-based index of a node is equal to its random value index, indicating that random values are ordered.

4.3 Improved Ordering Algorithm

In this algorithm, each node i searches its own view $\mathcal{N}_i$ for misplaced neighbors. Then, one of them is chosen to swap random value with. This process is repeated until there is no global disorder. In this version of the algorithm, we provide each node with the capability of measuring locally the disorder. This leads to a new heuristic for each node to determine the neighbor to exchange with which decreases most the disorder.

The proposed technique attempts to decrease the global disorder in each exchange as much as possible via selecting the neighbor from the view that minimizes the local disorder (or, equivalently, maximizes the order *gain*) as defined below.

For a node i to evaluate the gain of exchanging with a node j of its current view $\mathcal{N}_i$, we define its *local disorder measure* (abbreviated *LDM*$_i$). Let $LA.sequence_i$ and $LR.sequence_i$ be the local attribute sequence and the local random sequence of node i, respectively. These sequences are computed locally by i using the information $\mathcal{N}_i \cup \{i\}$. Similarly to $A.sequence$ and $R.sequence$, these are the sequences of neighbors where each node is ordered according to its attribute value and random number, respectively. Let, for any $j \in \mathcal{N}_i \cup \{i\}$, $\ell\rho_j(t)$ and $\ell\alpha_j(t)$ be the indices of r_j and a_j in sequences $LR.sequence_i$ and $LA.sequence_i$, respectively, at time (t). At any time t, the local disorder measure of node i is defined as:

$$LDM_i(t) = \frac{1}{c+1}\sum_{j \in \mathcal{N}_i(t) \cup \{i\}} (\ell\alpha_j(t) - \ell\rho_j(t))^2.$$

We denote by $G_{i,j}(t+1)$ the reduction on this measure that i obtains after exchanging its random value with node j between time t and $t+1$. We define it as:

$$\begin{aligned} G_{i,j}(t+1) &= LDM_i(t) - LDM_i(t+1), \\ G_{i,j}(t+1) &= \frac{(\ell\alpha_i(t) - \ell\rho_i(t))^2 + (\ell\alpha_j(t) - \ell\rho_j(t))^2 - (\ell\alpha_i(t) - \ell\rho_j(t))^2 - (\ell\alpha_j(t) - \ell\rho_i(t))^2}{c+1} \end{aligned} \quad (1)$$

The heuristic used chooses for node i the misplaced neighbor j that maximizes $G_{i,j}(t+1)$.

Variable	Description
j	the identifier of the neighbor
t_j	the age of the neighbor
a_j	the attribute value of the neighbor
r_j	the random value of the neighbor

Table 1: The array corresponding to the view entry of the neighbor j ($j \in \mathcal{N}_i$).

4.3.1 Sampling Uniformly at Random

The algorithm relies on the fact that potential misplaced nodes are found so that they can swap their random numbers thereby increasing order. If the global disorder is high, it is very likely that any given node has misplaced neighbors in its view to exchange with. Nevertheless, as the system gets ordered, it becomes more unlikely for a node i to have misplaced neighbors. In this stage the way the view is composed plays a crucial role: if fresh samples from the network are not available, convergence can be slower than optimal.

Several protocols may be used to provide a random and dynamic sampling in a peer to peer system such as Newscast [12], Cyclon [18] or Lpbcast [9]. They differ mainly by their *closeness* to the uniform random sampling of the neighbors and the way they handle churn. In this report, we chose to use a variant of the Cyclon protocol to construct and update the views [7], as it is reportedly the best approach to achieve a uniform random neighbor set for all nodes.

4.3.2 Description of the Algorithm

The algorithm is presented in Figure 2. The active thread at node i runs the membership (gossiping) procedure (recompute-view$()_i$) and the exchange of random values periodically. As motivated above, the membership procedure, specified in Figure 3, is similar to the Cyclon algorithm: each node i maintains a view $\mathcal{N}_i$ containing one entry per neighbor. The entry of a neighbor j corresponds to a tuple presented in Table 1. Node i copies its view, selects the oldest neighbor j of its view, removes the entry e_j of j from the copy of its view, and finally sends the resulting copy to j. When j receives the view, j sends its own view back to i discarding possible pointers to i, and i and j update their view with the one they receive. This variant of Cyclon, as opposed to the original version, exchanges all entries of the view at each step.

The algorithm for exchanging random values from node i starts by measuring the ordering that can be gained by swapping with each neighbor (Lines 4–8). Then, i chooses the neighbor $j \in \mathcal{N}_i$ that maximizes gain $G_{i,k}$ for any of its neighbor k. Formally, i finds $j \in \mathcal{N}_i$ such that for any $k \in \mathcal{N}_i$, we have

$$G_{i,j}(t+1) \quad \geq \quad G_{i,k}(t+1). \tag{2}$$

Using the definition of $G_{i,j}$ in Equation (1), Equation (2) is equivalent to

$$\ell\alpha_i(t)\ell\rho_j(t) + \ell\alpha_j(t)\ell\rho_i(t) - \ell\alpha_j(t)\ell\rho_j(t) \quad \geq \quad \ell\alpha_i(t)\ell\rho_k(t) + \ell\alpha_k(t)\ell\rho_i(t) - \ell\alpha_k(t)\ell\rho_k(t).$$

Initial state of node i

(1) $period_i$, initially set to a constant;
r_i, a random value chosen in $(0, 1]$; a_i, the attribute value;
$slice_i \leftarrow \perp$, the slice i belongs to; $\mathcal{N}_i$, the view;
$gain_{j'}$, a real value indicating the gain achieved by exchanging with j';
$gain\text{-}max = 0$, a real.

Active thread at node i

(2) $\mathsf{wait}(period_i)$
(3) $\mathsf{recompute\text{-}view}()_i$
(4) **for** $j' \in \mathcal{N}_i$
(5) **if** $gain_{j'} \geq gain\text{-}max$ **then**
(6) $gain\text{-}max \leftarrow gain_{j'}$
(7) $j \leftarrow j'$
(8) **end for**
(9) $\mathsf{send}(\mathsf{REQ}, r_i, a_i)$ to j
(10) $\mathsf{recv}(\mathsf{ACK}, r'_j)$ from j
(11) $r_j \leftarrow r'_j$
(12) **if** $(a_j - a_i)(r_j - r_i) < 0$ **then**
(13) $r_i \leftarrow r_j$
(14) $slice_i \leftarrow \mathcal{S}_{l,u}$ such that $l < r_i \leq u$

Passive thread at node i **activated upon reception**

(15) $\mathsf{recv}(\mathsf{REQ}, r_j, a_j)$ from j
(16) $\mathsf{send}(\mathsf{ACK}, r_i)$ to j
(17) **if** $(a_j - a_i)(r_j - r_i) < 0$ **then**
(18) $r_i \leftarrow r_j$
(19) $slice_i \leftarrow \mathcal{S}_{l,u}$ such that $l < r_i \leq u$

Figure 2: Dynamic ordering by exchange of random values.

Active thread at node i
(1) **for** $j' \in \mathcal{N}_i$ **do** $t_{j'} \leftarrow t_{j'} + 1$ **end for**
(2) $j \leftarrow j'' : t_{j''} = \mathsf{max}_{j' \in \mathcal{N}_i}(t_{j'})$
(3) $\mathsf{send}(\mathsf{REQ}', \mathcal{N}_i \setminus \{e_j\} \cup \{\langle i, 0, a_i, r_i \rangle\})$ to j
(4) $\mathsf{recv}(\mathsf{ACK}', \mathcal{N}_j)$ from j
(5) $\mathit{duplicated\text{-}entries} = \{e : e.id \in \mathcal{N}_j \cap \mathcal{N}_i\}$
(6) $\mathcal{N}_i \leftarrow \mathcal{N}_i \cup (\mathcal{N}_j \setminus \mathit{duplicated\text{-}entries} \setminus \{e_i\})$

Passive thread at node i **activated upon reception**
(7) $\mathsf{recv}(\mathsf{REQ}', \mathcal{N}_j)$ from j
(8) $\mathsf{send}(\mathsf{ACK}', \mathcal{N}_i)$ to j
(9) $\mathit{duplicated\text{-}entries} = \{e \in \mathcal{N}_j : e.id \in \mathcal{N}_j \cap \mathcal{N}_i\}$
(10) $\mathcal{N}_i \leftarrow \mathcal{N}_i \cup (\mathcal{N}_j \setminus \mathit{duplicated\text{-}entries})$

Figure 3: $\mathsf{recompute\text{-}view}()$: procedure used to update the view based on a simple variant of the Cyclon algorithm.

In Figure 2 of node i, we refer to $gain_j$ as the value of $\ell\alpha_i(t)\ell\rho_j(t) + \ell\alpha_j(t)\ell\rho_i(t) - \ell\alpha_j(t)\ell\rho_j(t)$.

From this point on, i exchanges its random value r_i with the random value r_j of node j (Line 11). The passive threads are executed upon reception of a message. In Figure 2, when j receives the random value r_i of node i, it sends back its own random value r_j for the exchange to occur (Lines 15–16). Observe that the attribute value of i is also sent to j, so that j can check if it is correct to exchange before updating its own random number (Lines 17–18). Node i does not need to receive attribute value a_j of j, since i already has this information in its view and the attribute value of a node never changes over time.

4.4 Analysis of Slice Misplacement

In mod-JK, as in JK, the current random number r_i of a node i determines the slice s_i of the node. The objective of both algorithms is to reduce the global disorder as quickly as possible. Algorithm mod-JK consists in choosing one neighbor among the possible neighbors that would have been chosen in JK, plus the GDM of JK has been shown to fit an exponential decrease [10]. Consequently mod-JK experiences also an exponential decrease of the global disorder. Eventually, JK and mod-JK ensure that the disorder has fully disappeared. However, the accuracy of the slices heavily depends on the uniformity of the random value spread between 0 and 1. It may happen, that the distribution of the random values is such that some peers decide upon a wrong slice. Even more problematic is the fact that this situation is unrecoverable unless a new random value is drawn for all nodes. This may be considered as an inherent limitation of the approach. For example, consider a system of size 2, where nodes 1 and 2 have the random values $r_1 = 0.1$, $r_2 = 0.4$. If we are interested in creating two slices of equal size, the first slice will be of size 2 and the second of size zero, even after perfect ordering of the random values.

Therefore, an important step is to characterize the inaccuracy of the uniform distribution to access the potential impact on the slice assignment resulting from the fact that uniformly generated random

numbers are not distributed perfectly evenly throughout the domain. First of all, consider a slice S_p of length p. In a network of n nodes, the number of nodes that will fall into this slice is a random variable X with a binomial distribution with parameters n and p. The standard deviation of X is therefore $\sqrt{np(1-p)}$. This means that the relative proportional expected difference from the mean (i.e., np) can be approximated as $\sqrt{(1-p)/(np)}$, which is very large if p is small, in fact, goes to infinity as p tends to zero, although a very large n compensates for this effect. For a "normal" value of p, and a reasonably large network, the variance is very low however.

To stay with this random variable, the following result bounds, with high probability, its deviation from its mean.

Lemma 4.1. *For any $\beta \in (0,1]$, a slice S_p of length $p \in (0,1]$ has a number of peers $X \in [(1-\beta)np, (1+\beta)np]$ with probability at least $1-\epsilon$ as long as $p \geq \frac{3}{\beta^2 n}\ln(2/\epsilon)$.*

Proof. The way nodes choose their random number is like drawing n times, with replacement and independently uniformly at random, a value in the interval $(0,1]$. Let $X_1, ..., X_n$ be the n corresponding independent identically distributed random variables such that:

$$\begin{cases} X_i & = 1 \text{ if the value drawn by node } i \text{ belongs to } S_p \text{ and} \\ X_i & = 0 \text{ otherwise.} \end{cases}$$

We denote $X = \sum_{i=1}^{n} X_i$ the number of elements of interval S_p drawn among the n drawings. The expectation of X is np. From now on we compute the probability that a bounded portion of the expected elements are misplaced. Two Chernoff bounds [14] give:

$$\left.\begin{array}{ll} \Pr[X \geq (1+\beta)np] & \leq e^{-\frac{\beta^2 np}{3}} \\ \Pr[X \leq (1-\beta)np] & \leq e^{-\frac{\beta^2 np}{2}} \end{array}\right\} \Rightarrow \Pr[|X - np| \geq \beta np] \leq 2e^{-\frac{\beta^2 np}{3}},$$

with $0 < \beta \leq 1$. That is, the probability that more than (β time the number expected) elements are misplaced regarding to interval S_p is bounded by $2e^{-\frac{\beta^2 np}{3}}$. We want this to be at most ϵ. This yields the result. □

To measure the effect discussed above during the simulation experiments, we introduce the slice disorder measure (SDM) as the sum over all nodes i of the distance between the slice i actually belongs to and the slice i believes it belongs to. For example (in the case where all slices have the same size), if node i belongs to the 1st slice (according to its attribute value) while it thinks it belongs to the 3rd slice (according to its rank estimate) then the distance for node i is $|1-3| = 2$. Formally, for any node i, let S_{u_i,l_i} be the actual correct slice of node i and let $S_{\hat{u_i},\hat{l_i}}(t)$ be the slice i estimates as its slice at time t. The slice disorder measure is defined as:

$$SDM(t) = \sum_i \frac{1}{u_i - l_i} \left| \frac{u_i + l_i}{2} - \frac{\hat{u_i} + \hat{l_i}}{2} \right|.$$

$SDM(t)$ is minimal (equals 0) if for all nodes i, we have $S_{\hat{u_i},\hat{l_i}}(t) = S_{u_i,l_i}$.

In fact, it is simple to show that, in general, the probability of dividing n peers into two slices of the same size is less than $\sqrt{2/n\pi}$. This value is very small even for moderate values of n. Hence, it is highly possible that the random number distribution does not lead to a perfect division into slices.

4.5 Simulation Results

We present simulation results using PeerSim [11], using a simplified cycle-based simulation model, where all messages exchanges are atomic, so messages never overlap. First, we compare the performance of the two algorithms: JK and mod-JK. Second, we study the impact of concurrency that is ignored by the cycle-based simulations.

4.5.1 Performance Comparison

We compare the time taken by these algorithms to sort the random values according to the attribute values (i.e., the node with the jth largest attribute value of the system value obtains the jth random value). In order to evaluate the convergence speed of each algorithm, we use the slice disorder measure as defined in Section 4.4.

We simulated 10^4 participants in 100 equally sized slices (when unspecified), each with a view size $c = 20$. Figure 4(a) illustrates the difference between the global disorder measure and the slice disorder measure while Figure 4(b) presents the evolution of the slice disorder measure over time for JK, and mod-JK.

Figure 4(a) shows the different speed at which the global disorder measure and the slice disorder measure converge. When values are sufficiently large, the GDM and SDM seem tightly related: if GDM increases then SDM increases too. Conversely, there is a significant difference between the GDM and SDM when the values are relatively low: the GDM reaches 0 while the SDM is lower bounded by a positive value. This is because the algorithm does lead to a totally ordered set of nodes, while it still does not associate each node with its correct slice. Consequently the GDM is not sufficient to rightly estimate the performance of our algorithms.

Figure 4(b) shows the slice disorder measure to compare the convergence speed of our algorithm to that of JK with 10 equally sized slices. Our algorithm converges significantly faster than JK. Note that none of the algorithm reaches zero SDM, since they are both based on the same idea of sorting randomly generated values. Besides, since they both used an identical set of randomly generated values, both converge to the same SDM.

4.5.2 Concurrency

The simulations are cycle-based and at each cycle an algorithm step is done atomically so that no other execution is concurrent. More precisely, the algorithms are simulated such that in each cycle, each node updates its view before sending its random value or its attribute value. Given this implementation, the cycle-based simulator does not allow us to realistically simulate concurrency, and a drawback is that view is up-to-date when a message is sent. In the following we artificially introduce concurrency (so that view might be out-of-date) into the simulator and show that it has only a slight impact on the convergence speed.

Introducing concurrency might result in some problems because of the potential staleness of views: unsuccessful swaps due to useless messages. Technically, the view of node i might indicate that j has a random value r while this value is no longer up-to-date. This happens if i has lastly updated its view before j swapped its random value with another j'. Moreover, due to asynchrony,

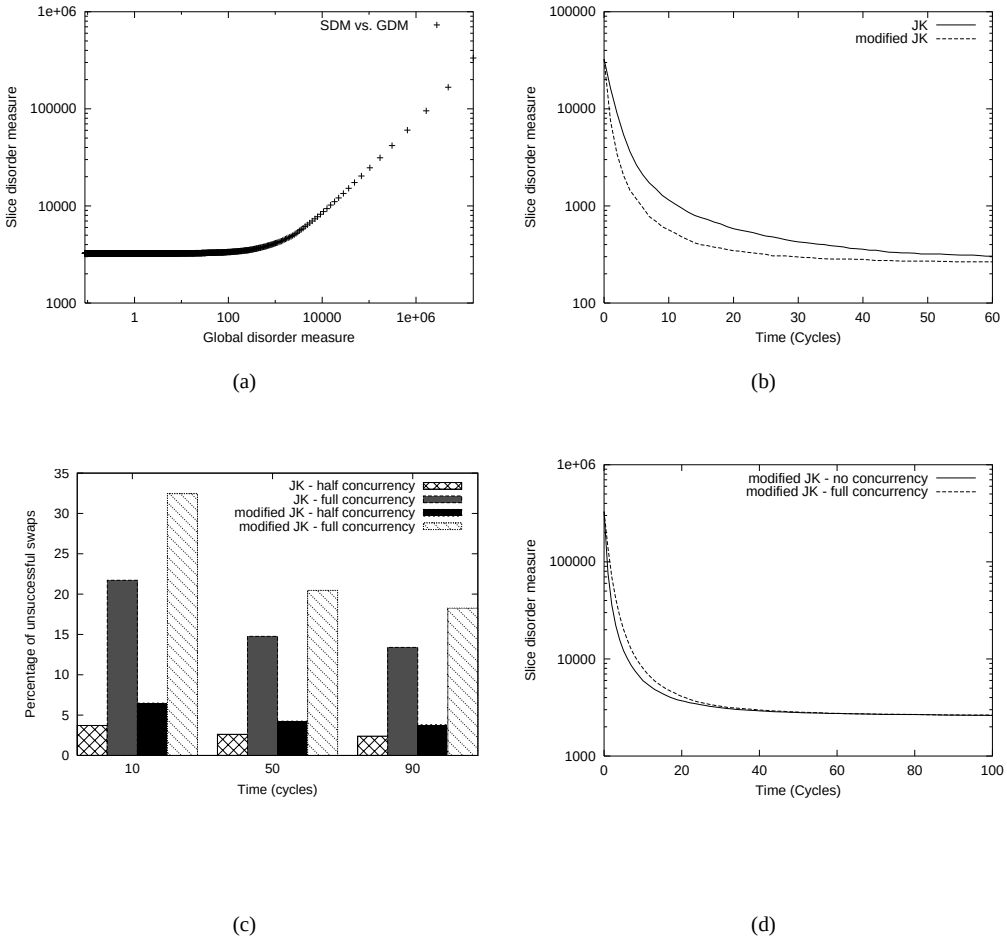

Figure 4: (a) Evolution of the global disorder measure over time. (b) Slice disorder measure over time. (c) Percentage of unsuccessful swaps in the ordering algorithms. (d) Convergence speed under high concurrency.

it could happen that by the time a message is received this message has become useless. Assume that node i sends its random value r_i to j in order to obtain r_j at time t and j receives it by time $t + \delta$. With no loss of generality assume $r_i > r_j$. Then if j swaps its random value with j' such that $r'_j > r_i$ between time t and $t + \delta$, then the message of i becomes *useless* and the expected swap does not occur (we call this an *unsuccessful swap*).

Figure 4(d) indicates the impact of concurrent message exchange on the convergence speed while Figure 4(c) shows the amount of useless messages that are sent. Now, we explain how the concurrency is simulated. Let the *overlapping messages* be a set of messages that mutually overlap: it exists, for any couple of overlapping messages, at least one instant at which they are both in-transit. For each algorithm we simulated *(i)* full concurrency: in a given cycle, all messages are overlapping messages; and *(ii)* half concurrency: in a given cycle, each message is an overlapping message with probability $\frac{1}{2}$. Generally, we see that increasing the concurrency increases the number of useless messages. Moreover, in the modified version of JK, more messages are ignored than in the original JK algorithm. This is due to the fact that some nodes (the most misplaced ones) are more likely targeted which increases the number of concurrent messages arriving at the same nodes. Since a node i ignored more likely a message when it receives more messages during the same cycle, it comes out that concentrating message sending at some targets increases the number of useless messages.

Figure 4(d) compares the convergence speed under full concurrency and no concurrency. We omit the curve of half-concurrency since it would have been similar to the two other curves. Full-concurrency impacts on the convergence speed very slightly.

5 Dynamic Ranking by Sampling of Attribute Values

In this section we propose an alternative algorithm for the distributed slicing problem. This algorithm circumvents some of the problems identified in the previous approach by continuously ranking nodes based on observing attribute value information. Random values no longer play a role, so non-perfect uniformity in the random value distribution is no longer a problem. Besides, this algorithm is not sensitive to churn even if it is correlated with attribute values.

In the remaining part of the report we refer to this new algorithm as the ranking algorithm while referring to JK and mod-JK as the ordering algorithms. Here, we elaborate on the drawbacks arising from the ordering algorithms relying on the use of random values that are solved by the ranking approach.

Impact of attribute correlated with dynamics. As already mentioned, the ordering algorithms rely on the fact that random values are uniformly distributed. However, if the attribute values are not constant but correlated with the dynamic behavior of the system, the distribution of random values may change from uniform to skewed quickly. For instance, assume that each node maintains an attribute value that represents its own lifetime. Although the algorithm is able to quickly sort random values, so nodes with small lifetime will obtain the small random values, it is more likely that these nodes leave the system sooner than other nodes. This results in a higher concentration of high random values and a large population of the nodes wrongly estimate themselves as being part of the higher slices.

Inaccurate slice assignments. As discussed in previous sections in detail, slice assignments will typically be imperfect even when the random values are perfectly ordered. Since the ranking approach does not rely on ordering random nodes, this problem is not raised: the algorithm guarantees eventually perfect assignment in a static environment.

Concurrency side-effect. In the previous ordering algorithms, a non negligible amount of messages are sent unnecessarily. The concurrency of messages has a drastic effect on the number of useless messages as shown previously, slowing down convergence. In the ranking algorithm concurrency has no impact on convergence speed because all received messages are taken in account. This is because the information encapsulated in a message (the attribute value of a node) is guaranteed to be up to date, as long as the attribute values are constant, or at least change slowly.

5.1 Ranking Algorithm Specification

The pseudocode of the ranking algorithm is presented in Figure 5. As opposed to the ordering algorithm of the previous section, the ranking algorithm does not assign random initial unalterable values as candidate ranks. Instead, the ranking algorithm improves its rank estimate each time a new message is received.

The ranking algorithm works as follows. Periodically each node i updates its view $\mathcal{N}_i$ following an underlying protocol that provides a uniform random sample (Line 3); later, we simulate the algorithm using the variant of Cyclon protocol presented in Section 4.3.2. Node i computes its rank estimate (and hence its slice) by comparing the attribute value of its neighbors to its own attribute value. This estimate is set to the ratio of the number of nodes with a lower attribute value that i has seen over the total number of nodes i has seen (Line 15). Node i looks at the normalized rank estimate of all its neighbors. Then, i selects the node j_1 closest to a slice boundary (according to the rank estimates of its neighbors). Node i selects also a random neighbor j_2 among its view (Line 12). When those two nodes are selected, i sends an update message, denoted by a flag UPD, to j_1 and j_2 containing its attribute value (Line 13–14).

The reason why a node close to the slice boundary is selected as one of the contacts is that such nodes need more samples to accurately determine which slice they belong to (subsection 5.2 shows this point). This technique introduces a bias towards them, so they receive more messages.

Upon reception of a message from node i, the passive threads of j_1 and j_2 are activated so that j_1 and j_2 compute their new rank estimate r_{j_1} and r_{j_2}. The estimate of the slice a node belongs to, follows the computation of the rank estimate. Messages are not replied, communication is one-way, resulting in identical message complexity to JK and mod-JK.

5.2 Theoretical Analysis

The following Theorem shows a lower bound on the probability for a node i to accurately estimate the slice it belongs to. This probability depends not only on the number of attribute exchanges but also on the rank estimate of i.

Initial state of node i

(1) $period_i$, initially set to a constant; r_i, a value in $(0, 1]$; a_i, the attribute value; b, the closest slice boundary to node i; g_i, the counter of encountered attribute values; l_i, the counter of lower attribute values; $slice_i \leftarrow \perp$; $\mathcal{N}_i$, the view.

Active thread at node i

(2) $\mathsf{wait}(period_i)$
(3) $\mathsf{recompute\text{-}view}()_i$
(4) $dist\text{-}min \leftarrow \infty$
(5) **for** $j' \in \mathcal{N}_i$
(6) $\quad g_i \leftarrow g_i + 1$
(7) $\quad$ **if** $a_{j'} \leq a_i$ **then** $\ell_i \leftarrow \ell_i + 1$
(8) $\quad$ **if** $\mathsf{dist}(a_{j'}, b) < dist\text{-}min$ **then**
(9) $\quad\quad dist\text{-}min \leftarrow \mathsf{dist}(a_{j'}, b)$
(10) $\quad\quad j_1 \leftarrow j'$
(11) **end for**
(12) Let j_2 be a random node of $\mathcal{N}_i$
(13) $\mathsf{send}(\mathsf{UPD}, a_i)$ to j_1
(14) $\mathsf{send}(\mathsf{UPD}, a_i)$ to j_2
(15) $r_i \leftarrow \ell_i / g_i$
(16) $slice \leftarrow \mathcal{S}_{l,u}$ such that $l < r_i \leq u$

Passive thread at node i **activated upon reception**

(17) $\mathsf{recv}(\mathsf{UPD}, a_j)$ from j
(18) **if** $a_j \leq a_i$ **then** $\ell_i \leftarrow \ell_i + 1$
(19) $g_i \leftarrow g_i + 1$
(20) $r_i \leftarrow \ell_i / g_i$
(21) $slice \leftarrow \mathcal{S}_{l,u}$ such that $l < r_i \leq u$

Figure 5: Dynamic ranking by exchange of attribute values.

Theorem 5.1. *Let p be the normalized rank of i and let $\hat{p}$ be its estimate. For node i to exactly estimate its slice with confidence coefficient of $100(1-\alpha)\%$, the number of messages i must receive is:*

$$\left(Z_{\frac{\alpha}{2}}\frac{\sqrt{\hat{p}(1-\hat{p})}}{d}\right)^2,$$

where d is the distance between the rank estimate of i and the closest slice boundary, and $Z_{\frac{\alpha}{2}}$ represents the endpoints of the confidence interval.

Proof. Each time a node receives a message, it checks whether or not the attribute value is larger or lower than its own. Let $X_1, ..., X_k$ be k $(k > 0)$ independent identically distributed random variables described as follows. $X_j = 1$ with probability $\frac{i}{n} = p$ (indicating that the attribute value is lower) and $j \in \{1, ..., k\}$, otherwise $X_j = 0$ (indicating the attribute value is larger). By the central limit theorem, we assume $k > 30$ and we approximate the distribution of $X = \sum_{j=1}^{k} X_j$ as the normal distribution. We estimate X by $\hat{X} = \sum_{j=1}^{k} \hat{X}_j$ and p by $\hat{p} = \frac{\hat{X}}{k}$.

We want a confidence coefficient with value $1-\alpha$. Let Φ be the standard normal distribution function, and let $Z_{\frac{\alpha}{2}}$ be $\Phi^{-1}(1-\frac{\alpha}{2})$. Now, by the Wald large-sample normal test in the binomial case, where the standard deviation of $\hat{p}$ is $\sigma(\hat{p}) = \frac{\sqrt{\hat{p}(1-\hat{p})}}{\sqrt{k}}$, we have:

$$\begin{aligned} \left|\frac{\hat{p}-p}{\sigma(\hat{p})}\right| &\leq Z_{\frac{\alpha}{2}} \\ \hat{p} - Z_{\frac{\alpha}{2}}\sigma(\hat{p}) \quad &\leq p \quad \leq \hat{p} + Z_{\frac{\alpha}{2}}\sigma(\hat{p}). \end{aligned}$$

Next, assume that $\hat{p}$ falls into the slice $S_{l,u}$, with l and u its lower and upper boundaries, respectively. Then, as long as $\hat{p} - Z_{\frac{\alpha}{2}}\sqrt{\frac{\hat{p}(1-\hat{p})}{k}} > l$ and $\hat{p} + Z_{\frac{\alpha}{2}}\sqrt{\frac{\hat{p}(1-\hat{p})}{k}} \leq u$, the slice estimate is exact with a confidence coefficient of $100(1-\alpha)\%$. Let $d = \min(\hat{p}-l, u-\hat{p})$, then we need

$$\begin{aligned} d &\geq Z_{\frac{\alpha}{2}}\sqrt{\frac{\hat{p}(1-\hat{p})}{k}}, \\ k &\geq \left(Z_{\frac{\alpha}{2}}\frac{\sqrt{\hat{p}(1-\hat{p})}}{d}\right)^2. \end{aligned}$$

□

To conclude, under reasonable assumptions all node estimate its slice with confidence coefficient $100(1-\alpha)\%$, after a finite number of message receipts. Moreover a node closer to the slice boundary needs more messages than a node far from the boundary.

5.3 Simulation Results

This section evaluates the ranking algorithm by focusing on three different aspects. First, the performance of the ranking algorithm is compared to the performance of the ordering algorithm[2] in a

[2]We omit comparison with JK since the performance obtained with mod-JK are either similar or better.

large-scale system where the distribution of attribute values does not vary over time. Second, we investigate if sufficient uniformity is achievable in reality using a dedicated protocol. Third, the ranking algorithm and ordering algorithm are compared in a dynamic system where the distribution of attribute values may change.

For this purpose, we ran two simulations, one for each algorithms. The system contains (initially) 10^4 nodes and each view contains 10 uniformly drawn random nodes and is updated in each cycle. The number of slices is 100, and we present the evolution of the slice disorder measure over time.

5.3.1 Performance Comparison in the Static Case

Figure 6(a) compares the ranking algorithm to the ordering algorithm while the distribution of attribute values do not change over time (varying distribution is simulated below).

The difference between the ordering algorithm and the ranking algorithm indicates that the ranking algorithm gives a more precise result (in terms of node to slice assignments) than the ordering algorithm. More importantly, the slice disorder measure obtained by the ordering algorithm is lower bounded while the one of the ranking algorithm is not. Consequently, this simulation shows that the ordering algorithm might fail in slicing the system while the ranking algorithm keeps improving its accuracy over time.

5.3.2 Feasibility of the Ranking Algorithm

Figure 6(b) shows that the ranking algorithm does not need artificial uniform drawing of neighbors. Indeed, an underlying view management protocol might lead to similar performance results. In the presented simulation we used an artificial protocol, drawing neighbors randomly at uniform in each cycle of the algorithm execution, and the variant of the Cyclon [18] view management protocol presented above. Those underlying protocols are distinguished on the figure using terms ”uniform” (for the former one) and ”views” (for the later one). As said previously, the Cyclon protocol consists of exchanging views between neighbors such that the communication graph produced shares similarities with a random graph. This figure shows that both cases give very similar results. The SDM legend is on the right-handed vertical axis while the left-handed vertical axis indicates what percentage the SDM difference represents over the total SDM value. At any time during the simulation (and for both type of algorithms) its value remains within plus or minus 7%. The two SDM curves of the ranking algorithm almost overlap. Consequently, the ranking algorithm and the variant of Cyclon presented in subection 4.3.2 achieve very similar result.

To conclude, the variant of Cyclon algorithm presented in the previous section can be used with the ranking algorithm to provide the shuffling of views.

5.3.3 Performance Comparison in the Dynamic Case

In Figure 6(c) each of the two curves represents the slice disorder measure obtained over time using the ordering algorithm and the ranking algorithm respectively. We simulate the churn such that 0.1% of nodes leave and 0.1% of the nodes join in each cycle during the 200 first cycles. We observe

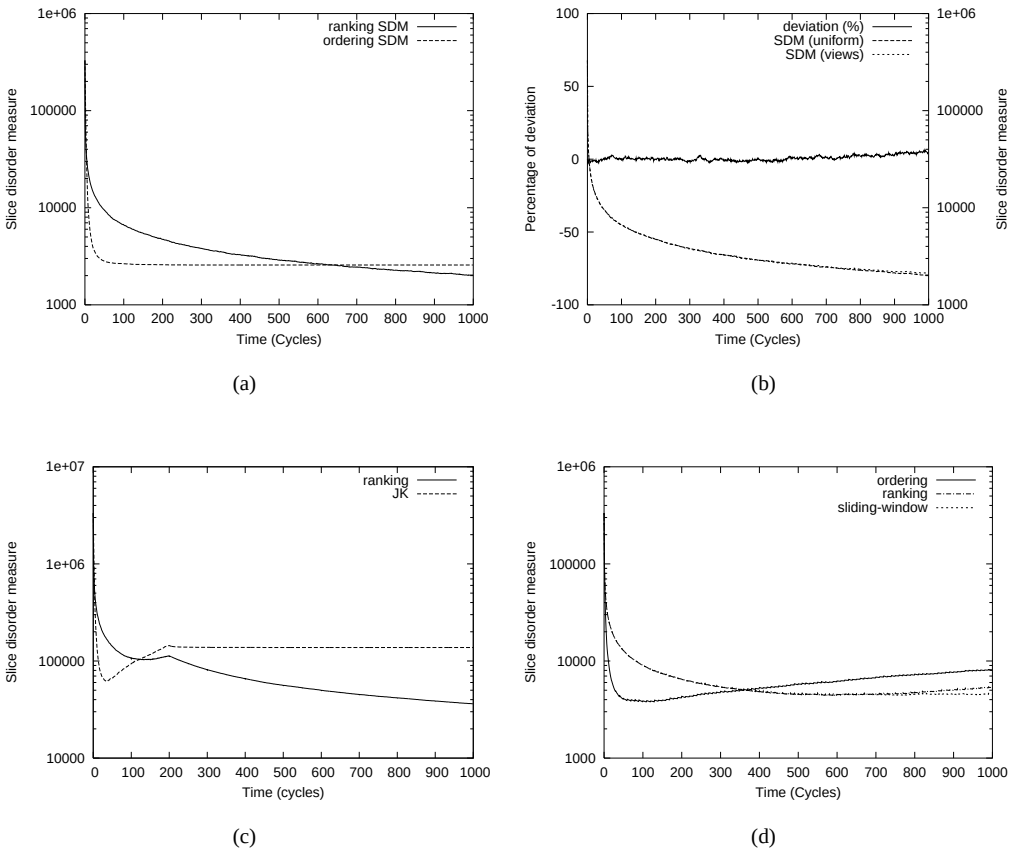

Figure 6: (a) Comparing performance of the ordering algorithm and the ranking algorithm (static case). (b) Comparing the ranking algorithm on top of a uniform drawing or a Cyclon-like protocol. (c) Effect of dynamics burst on the convergence of the ordering algorithm and the ranking algorithm. (d) Effect of a low and regular churn on the convergence of the ordering algorithm and the ranking algorithm.

how the SDM converges. The churn is reasonably and pessimistically tuned compared to recent experimental evaluations [17] of the session duration in three well-known P2P systems.[3]

The distribution of the churn is correlated to the attribute value of the nodes. The leaving nodes are the nodes with the lowest attribute values while the entering nodes have higher attribute values than all nodes already in the system. The parameter choices are motivated by the need of simulating a system in which the attribute value corresponds to the session duration of nodes, for example.

The churn introduces a significant disorder in the system which counters the fast decrease. When, the churn stops, the ranking algorithm readapts well the slice assignments: the SDM starts decreasing again. However, in the ordering algorithm, the convergence of SDM gets stuck. This leads to a poor slice assignment accuracy.

In Figure 6(d), each of the two curves represent the slice disorder measure obtained over time using the ordering algorithm, the ranking algorithm, and a modified version of the ranking algorithm using attribute values recorded in a sliding-window, respectively. (The simulation obtained using sliding windows is described in the next subsection.) The churn is diminished and made more regular than in the previous simulation such that 0.1% of nodes leave and 0.1% of nodes join every 10 cycles.

The curves fits a fast decrease (superlinear in the number of cycles) at the beginning of the simulation. At first cycles, the ordering gain is significant making the impact of churn negligible. This phenomenon is due to the fact that SDM decreases rapidly when the system is fully disordered. Later on, however, the decrease slope diminishes and the churn effect reduces the amount of nodes with a low attribute value while increasing the amount of nodes with a large attribute value. This unbalance leads to a messy slice assignment, that is, each node must quickly find its new slice to prevent the SDM from increasing. In the ordering algorithm the SDM starts increasing from cycle 120. Conversely, with the ranking algorithm the SDM starts increasing not earlier than at cycle 730. Moreover the increase slope is much larger in the former algorithm than in the latter one.

Even though the performance of the ranking algorithm are really significant, its adaptiveness to churn is not surprising. Unlike the ordering algorithm, the ranking one keeps re-estimating the rank of each node depending on the attribute values present in the system. Since the churn increases the attribute values present in the system, nodes tend to receive more messages with higher attribute values and less messages with lower attribute values, which turns out to keep the SDM low, despite churn. Further on, we propose a solution based on sliding-window technique to limit the increase of the SDM in the ranking algorithm.

To conclude, the results show that when the churn is related to the attribute (e.g., attribute represents the session duration, uptime of a node), then the ranking algorithm is better suited than the ordering algorithm.

[3] In [17], roughly all nodes have left the system after 1 day while there are still 50% of nodes after 25 minutes. In our case, assuming that in average a cycle lasts one second would lead to more than 54% of leave in 9 minutes.

5.3.4 Sliding-window for Limiting the SDM Increase

In Figure 6(d), the ”sliding-window” curve presents a slightly modified version of the ranking algorithm that encompasses SDM increase due to churn correlated to attribute values. Here, we present this enrichment.

In Section 5, the ranking algorithm specifies that each node takes into account all received messages. More precisely, upon reception of a new message each node i re-computes immediately its rank estimate and the slice it thinks it belongs to without remembering the attribute values it has seen. Consequently the messages received long-time ago have as much importance as the fresh messages in the estimate of i. The drawback, as it appeared in Figure 6(d) of Section 4.5, is that if the attribute values are correlated to churn, then the precision of the algorithm might diminish.

To cope with this issue, the previous algorithm can be easily enriched in the following way. Upon reception of a message, each node i records an information about the attribute value received in a fixed-size ordered set of values. Say this set is a first-in first-out buffer such that only the most recent values remain. Right after having recorded this information, node i can re-compute its rank estimate and its slice estimate based on the most relevant piece of information (having discarded the irrelevant piece). Consequently, the estimate would rely only on fresh attribute values encountered so that the algorithm would be more tolerant to changes (e.g., dynamics or non-uniform evolution of attribute values). Of course, since the analysis (cf. Section 5.2) shows that nodes close to the slice boundary require a large number of attribute values for estimating precisely their estimates, it would be unaffordable to record all these last attribute values encountered due to space limitation.

Actually, the only necessary relevant information of a message is simply whether it contains a lower attribute value than the attribute value of i, or not. Consequently, a single bit per message would be sufficient to record the necessary information (e.g., adding a 1 meaning that the attribute value is lower, and 0 otherwise). Thus, even though a node i would require 10^4 messages to rightly estimate its slice (with high probability), node i simply needs to allocate an array of size $10^4/(8 * 1000) = 1,25$ kB.

As expected, Figure 6(d) shows that the sliding-window method applied to the ranking algorithm prevents its SDM from increasing. Consequently, at some point in time, the resulting slice assignment may become even more accurate.

6 Conclusion

6.1 Summary

Peer to peer systems may now be turned into general frameworks on top of which several applications might cohabit. To this end, allocating resources to applications, according to their needs require specific algorithms to partition the network in a relevant way. The ordered slicing algorithm proposed in [10] provided a first attempt to “slice” the network, taking into account the potential heterogeneity of nodes. This algorithm relies on each node drawing a random value uniformly and swapping continuously those random values, with candidate nodes, so that the order between attributes values

(reflecting the capabilities of nodes) and random ones match. Results from [10] have shown that slices can be maintained efficiently and in large-scale systems even in the presence of churn.

In this report, we first proposed an improvement over the initial sorting algorithm based on a judicious choice of candidate nodes to swap values. This is based on each node being able to estimate locally the potential decrease of the global disorder measure. We provided an analysis along with some simulation results showing that the convergence speed is significantly improved. We then identified two issues related to the use of static random values. The first one refers to the fact that slice assignment heavily depends on the degree of uniformity of the initial random value.

The second is related to the fact that once sorted along one attribute axis, the churn (or failures) might be correlated to the attribute, therefore leading to a unrecoverable skewed distribution of the random values resulting in a wrong slice assignment. Our second contribution is an algorithm enabling nodes to continuously re-estimate their rank relatively to other nodes based on their sampling of the network.

6.2 Perspective

This report used a variant of the Cyclon protocol to obtain quasi-uniform distribution of neighbors. There are various protocols that might be used for different purpose. For instance, Newscast can be used for its resilience to very high dynamics as in [10]. Some other protocols exist in the literature. Deciding exactly how to parameterize the underlying peer sampling service might be an interesting future direction.

Acknowledgment

We wish especially to thank Márk Jelasity for the fruitful discussions we had and the time he spent improving this report. We are also thankful to Spyros Voulgaris for having kindly shared his work on the Cyclon development. The work of A. Fernández and E. Jiménez was partially supported by the Spanish MEC under grants TIN2005-09198-C02-01, TIN2004-07474-C02-02, and TIN2004-07474-C02-01, and the Comunidad de Madrid under grant S-0505/TIC/0285. The work of A. Fernández was done while on leave at IRISA, supported by the Spanish MEC under grant PR-2006-0193.

References

[1] D. P. Anderson. Boinc: a system for public-resource computing and storage. In *Proceedings. Fifth IEEE/ACM International Workshop on Grid Computing*, pages 4–10, 2004.

[2] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *Symposium on Networked Systems Design and Implementation*, pages 253–266, 2004.

[3] R. Bhagwan, S. Savage, and G. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, pages 256–267, February 2003.

[4] M. Blum, R. Floyd, V. Pratt, R. Rivest, and R. Tarjan. Time bounds for selection. *J. Computer and System Sciences*, 7:448–461, 1972.

[5] David J. DeWitt, Jeffrey F. Naughton, and Donovan A. Schneider. Parallel sorting on a shared-nothing architecture using probabilistic splitting. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 280–291, 1991.

[6] Robert W. Floyd and Ronald L. Rivest. Expected time bounds for selection. *Commun. ACM*, 18(3):165–172, 1975.

[7] Konrad Iwanicki. Gossip-based dissemination of time. Master's thesis, Warsaw University - Vrije Universiteit Amsterdam, May 2005.

[8] B. Iyer, G. Ricard, and P. Varman. Percentile finding algorithm for multiple sorted runs. In *Proceedings of the 15th International Conference on Very Large Data Bases*, pages 135–144, August 1989.

[9] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, pages 79–98, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[10] Márk Jelasity and Anne-Marie Kermarrec. Ordered slicing of very large-scale overlay networks. In *Proc. of the Sixth IEEE International Conference on Peer-to-Peer Computing*, pages 117–124, 2006.

[11] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. A modular paradigm for building self-organizing peer-to-peer applications. In *Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering*, number 2977 in Lecture Notes in Artificial Intelligence, pages 265–282. Springer-Verlag, April 2004.

[12] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, August 2005.

[13] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregrate information. In *Proc. of 44th Annual IEEE Symposium of Foundations of Computer Science*, pages 482–491, 2003.

[14] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, England, 1995.

[15] J. Sacha, J. Dowling, R. Cunningham, and R. Meier. Using aggregation for adaptive super-peer discovery on the gradient topology. In *IEEE International Workshop on Self-Managed Networks, Systems and Services*, pages 77–90, 2006.

[16] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of SPIE (Proceedings of Multimedia Computing and Networking 2002, MMCN'02)*, volume 4673, pages 156–170, 2002.

[17] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Internet Measurement Conference*, pages 189–202, 2006.

[18] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, 2005.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399