Ordered Slicing of Very Large-Scale Overlay Networks*

Márk Jelasity†
University of Bologna, Italy
jelasity@cs.unibo.it

Anne-Marie Kermarrec
INRIA/IRISA, Rennes, France
akermarr@irisa.fr

Abstract

Recently there has been an increasing interest to harness the potential of P2P technology to design and build rich environments where services are provided and multiple applications can be supported in a flexible and dynamic manner. In such a context, resource assignment to services and applications is crucial. Current approaches require significant "manual-mode" operations and/or rely on centralized servers to maintain resource availability. Such approaches are neither scalable nor robust enough. Our contribution towards the solution of this problem is proposing and evaluating a gossip-based protocol to automatically partition the available nodes into "slices", also taking into account specific attributes of the nodes. These slices can be assigned to run services or applications in a fully self-organizing but controlled manner. The main advantages of the proposed protocol are extreme scalability and robustness. We present approximative theoretical models and extensive empirical analysis of the proposed protocol.

1. Introduction

Following the scale shift in distributed systems and their increasing dynamism, peer-to-peer overlay networks have imposed themselves as the key to build and maintain large-scale dynamic distributed systems. One important problem in the field of overlay networks is the design of infrastructures on which several applications might run together and share resources. Examples of such applications are Desktop-grid like computing platforms [1], and testbed platforms such as PlanetLab [2].

One key sub-problem is such environments is resource assignment to services and applications, and the definition of the resource itself. For example, in PlanetLab, the core concept is a slice, which refers to a virtualized network running over multiple physical nodes, and where each node can participate in multiple slices. Such slices are assigned to specific applications, sharing the platform. However, existing approaches are mostly manual and/or centralized. In contrast to this, we are interested in massively large scale and extremely dynamic networks, in which centralized slice assignment is not an option and where slices need not only to be assigned, but also *maintained*, to face constant churn.

In this paper, as a step towards a full self-organizing architecture, we focus on a well-defined problem: *ordered slicing*. Our objective is to create and maintain a partitioning of the network (we call the partitions *slices* in the remaining of the paper). This implies that slices are defined as subsets of the network, that is, each node belongs to exactly one slice at any given point in time. However, several such partitionings can be maintained in parallel. The ordered nature of the slicing means that specific attributes can be taken into account to partition the network: the partitioning is done along a fixed attribute of the nodes. For example, a service might require a slice composed of the top 20% of the nodes providing the largest bandwidth. Besides, we need to provide this top 20% constantly, even if the nodes in the top 20% constantly change due to churn or changing node properties.

Many metrics may be used to sort the nodes such as available resources (memory, bandwidth, computing power) or some specific behavioral pattern such as uptime. Note that slicing the network at random, and focusing only on the size of the slices is a special case of our ordered slicing protocol. We also note that the slice sizes are expressed as a percentage of the network, that is, if the network grows, slices grow accordingly.

We base our approach on a class of gossip-based protocols, that have been proven to be able to maintain connectivity in large-scale dynamic systems in the presence of high churn and other extreme failure scenarios [3]. This is achieved through maintaining a dynamic, pseudo-random overlay network. Due to their low cost, simplicity, scalability and robustness, these networks represent an ideal foundation to build our protocol upon.

The rest of the paper is organized as follows. In Section 2 we provide the problem statement and the sys-

*in *IEEE P2P*, pp. 117–124, 2006. This work was partially supported by the Future and Emerging Technologies unit of the European Commission through Project DELIS (IST-2002-001907).

†Also with RGAI, MTA SZTE, Szeged, Hungary.

tem model. In Section 3 we describe our gossip-based slicing protocol. An approximative theoretical model of our approach is presented in Section 4 and an extensive empirical analysis is presented in Section 5.

2. Problem Definition

2.1. System Model

We consider a network consisting of a large collection of *nodes* that are assigned unique identifiers (typically IP addresses) and that communicate through message exchanges. The network is highly dynamic; new nodes may join at any time, and existing nodes may leave, either voluntarily or by *crashing*. In the following, we limit our discussion to node crashes. Voluntary leaves are implemented as crashes: our protocols will not require a dedicated leave procedure, nor any failure detection. Successful delivery of messages happens without delay, however, messages may be dropped. Byzantine failures, with nodes behaving arbitrarily, are excluded from the present discussion.

We assume that nodes are connected through an existing physical routed network, such as the Internet, where every node can potentially communicate with every other node. To actually communicate, a node has to know the identifiers of a set of other nodes (its *neighbors*), for example, the IP address in the case of an IP network. This neighborhood relation over the nodes defines the topology of the *overlay network*. Given the large scale and the dynamism of our envisioned system, neighborhoods are typically limited to small subsets of the entire network. The neighbors of a node (and, thus, the overlay topology) may change dynamically over time.

2.2. The Ordered Slicing Problem

Intuitively, the ordered slicing problem asks for a partitioning of the nodes in the overlay network into groups (slices) in such a way, that the groups are ordered with respect to some given metric, such as the availability of a resource, or some other relevant property. For example, we might be interested in creating and maintaining a slice composed of the top 10% nodes according to available bandwidth, expected uptime, and so on. Note that creating a slice of a given size, populated with random nodes, is a special case where the metric is not taken into account, or, equivalently, assuming all nodes have the same value of the metric. Slice sizes are expressed as a percentage of the network size.

To define this problem, let N denote the network size and let each node i have an attribute, x_i. This value will typically measure the availability of some resource at node i. We assume that there exists a total ordering over the domain of the attributes values, so that the values in the network $(x_1, \ldots, x_N)$ can be ordered. Let us also assume that there is a *slice specification* that defines an ordered partitioning of the nodes. That is, the slice specification is a list of positive real numbers $s_1, \ldots, s_k$ such that $\sum s_i = 1$, that define slices $S_1, \ldots, S_k$, where the size of S_i is $s_i N$ and for all $i < j$, $a \in S_i$ and $b \in S_j$ we have $x_a \leq x_b$. We also assume that the slice specification is known at each node locally.

The problem is to automatically assign each node to slices in such a way that satisfies the slice specification, using only local message exchange with currently known neighbors. That is, we want each node to find out, which slice it belongs to, and, as a function of continuous changes in the network, maintain this assignment up-to-date.

The difficulty lies in the fact that the correct solution needs global information in that each nodes needs to calculate the number of nodes that preceed them in the total order according to the x attribute, and break ties whenever the number of preceeding nodes in not well defined (for example, if there are many identical attribute values in the network). Furthermore, the dynamism and failures in the system add extra difficulty, as this assignment needs to be continuously maintained in the face of a changing set of nodes.

As opposed to most traditional approaches that require eventual correctness provided there is no failure or change in the system for a sufficiently long time, we focus on a best effort approximation which is as close to the optimal solution as possible. In other words, instead of focusing on the worst case, we focus on optimizing the performance under normal dynamic operation.

Note that the solution we present in this paper can easily be extended to more general cases, such as when more independent attributes are involved, or when overlapping groups need to be maintained, and so on. However, the problem definition above allows us to keep the focus on the analysis of the key novel contributions introduced in this paper. We only sketch the possibilities of extensions in the conclusion.

3. A Gossip-based Approach

As mentioned previously, to let nodes decide locally whether they belong to a certain slice or not (expressed at a percentage of the whole size which is not known either), the key issue is to enable a node to approximate what percentage of nodes preceed it in the ordering according to the attribute value. There are at least two natural choices to implement this functionality. The first is through the application of protocols to calculate the ranking of the nodes in the ordering [6, 7]. However, known protocols are expensive and they are not suitable to *maintain* ranking information cheaply in the face of large scale and dynamism. The second is

to approximate the distribution of the attribute values and use this information to map any attribute value to an approximate ranking [9]. This approach however is not robust to skewed distributions and does not provide a sufficiently accurate information for the present purposes.

To deal with dynamism and large scale, we follow a third approach, which is based on the *sorting* of randomly generated numbers. The basic idea is that each node generates one uniform random number from a fixed interval, and subsequently the set of these random numbers are sorted "along the attribute values" with the help of the protocol we describe below. Sorting along the attribute values means that—via swapping the random numbers among a suitable sequence of pairs of nodes—we would like to achieve that the order of the random numbers reflects the order of the attribute values over the nodes.

After sorting, the node is able to make a judgment about its position in the sorting of the attribute values based on the random number it currently holds, because the distribution of the random numbers is known (that is, uniform from a fixed interval) and because the order of the random numbers reflect the order of the attribute values. For example, if the random numbers are drawn from $[0,1]$, then a node decides that it is in the first half of the sorting if, after sorting along the attribute values, it holds a value less than 0.5. Apart from being simple, this approach supports dynamism well, as all joining nodes can locally initialize their random number and subsequently participate in the sorting. Furthermore, the approach works independently of the distribution of the attribute values: they can even be identical at all nodes, in which case only the sizes of the slices are determined, but the nodes will be assigned to slices at random.

However, the sorting problem might seem equally difficult to our original problem. Our main contribution—apart from proposing the application of sorting—is a gossip-based sorting protocol that is simple to implement, incurs minimal costs and is efficient enough for the purposes of ordered slicing. The basic idea relies on a simple swapping of the random numbers between nodes. For example, let nodes i and j have attribute values $x_i = 10$ and $x_j = 20$, and random numbers $r_i = 0.8$ and $r_j = 0.1$. These nodes simply swap their random numbers in order to make them reflect the ordering of the attribute value. In order to make such pairs of nodes discover each other, we rely on a gossip-based algorithm.

Gossip-based algorithms have proven very efficient to create large-scale overlay networks that are extremely robust to failure and churn [3]. These algorithms are very simple: periodically, each node chooses a gossip target among its neighbors to exchange information with. The topology of the resulting overlay network depends on the target selection and the way the information exchange is processed.

```
1: loop
2:    wait(Δ_r)
3:    p ← random element from view
4:    buffer ← view ∪ {(myAddress,timestamp,x_q,r_q)}
5:    send buffer to p
6:    receive buffer_p from p
7:    view ← youngest c entries of buffer_p ∪ view
8:    i ← peer from view such that (x_i − x_q)(r_i − r_q) < 0
9:    send (x_q, r_q) to i
10:   r_q ← get r_i from i
11: end loop
```

(a) active thread at node q

```
1: loop
2:    receive buffer_q from q
3:    buffer ← view ∪ {(myAddress,timestamp,x_p,r_p)}
4:    send buffer to q
5:    view ← youngest c entries of buffer_q ∪ view
6: end loop
```

(b) passive thread at node p

Figure 1. The NEWSCAST sorting protocol. At node i, r_i is the currently held random value and x_i is the attribute value.

Specifically, our sorting protocol is based on the NEWSCAST protocol [5]. The purpose of NEWSCAST is to maintain a connected random overlay network in the face of dynamism, to implement the *peer sampling service* [3]. In addition, NEWSCAST can be used to spread "news", that is, information about participating nodes, that is useful only for a limited amount of time.

The basic idea behind the sorting algorithm is that each node will passively look for candidate peers to swap its random value with, in order to improve the sorting. These candidates are discovered using the constantly changing set of neighbors provided by the NEWSCAST protocol and the information (news) that is available over these neighbors.

The sorting protocol, derived from the NEWSCAST protocol and executed at each node, is shown in Figure 1. The VIEW is a set that contains the neighbors the node currently knows about. The size of the view is limited by parameter c. The view is composed of *node descriptors*, that contain, among other information, a timestamp of the creation of the descriptor. This timestamp is used to make decisions on which items to keep or throw away. Each node can have at most one descriptor in any view. When unifying two views, the freshest descriptor is kept for all nodes.

In the descriptor of node i, we store the attribute x_i and the random number r_i that was held by node i at the time of the creation of the descriptor. In line 8, the peer to exchange the random value with is selected based on this information. More specifically, in an attempt to in-

crease the probability of encountering a potential candidate for swapping, the node is looking for a peer that has a higher attribute value but a lower random value or vice versa. The rationale is that if this is not the case, it is guaranteed that the sorting cannot be improved by performing an exchange with the given peer.

Note that it is not guaranteed that a suitable peer exists in the view. In that case, no exchange is performed. Besides—since the information in the descriptor might be slightly outdated—it is possible that although the peer seems to be a suitable one for doing an exchange, in the meantime its random value has changed and is not actually suitable. In this case no exchange happens either.

Finally, although the protocol is not synchronous, it is often convenient to refer to *cycles* of the protocol. We define a cycle to be a time interval of Δ time units where Δ is the parameter of the protocol in Figure 1. Note that during a cycle, each node is updated twice on average: once when it sends its own message, and once on average when it receives a message.

4. Theoretical Models of the Basic Protocol

In this section, we present qualitative models of the protocol described in Figure 1. To qualitatively characterize its convergence speed, we will also apply theoretical results from a seemingly unrelated field: distributed gossip-based average calculation [4].

4.1. Basic Definitions

To simplify the notations (without restricting generality), we think of the network as a vector, ordered according to the attribute values x_i. We also assume here that the attribute values are stable (do not change) and that the set of nodes does not change either. That is, the order of the nodes in the imaginary sorted vector remains the same during the execution of the protocol. Finally, we also assume that the set of random numbers r_i held by the nodes does not change either. The assumptions above allow us to describe the state of the network simply by a permutation of the random values. Let this permutation at time t be denoted as $r_{i_1(t)},\ldots,r_{i_N(t)}$, expressing the fact that value $r_{i_j(t)}$ can be found at node $i_j(t)$ at time t. Furthermore, let the specific permutation $r_1,\ldots,r_N$.denote an ordering of the values, that is, if $i \leq j$ then $r_i \leq r_j$.

Clearly, in these notations, the goal of the protocol is to sort the values r_i through applying a series of binary permutations (swapping pairs of values). Let us assume that there is only one good sorting, that is, all the values are unique. Note that if some of the values are not unique, then the problem becomes easier; for example, if all the values r_i or x_i are the same, then all permutations are sorted and there is no problem to solve.

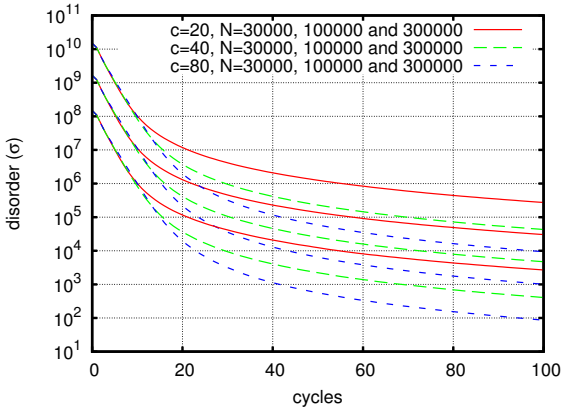

Figure 2. Disorder as a function of cycles. Curves for a fixed c completely overlap when normalized by N^2.

Using these notations and assumptions, we can introduce our measure of disorder to characterize the convergence of the protocol:

$$\sigma(t, r_{i_1(t)},\ldots,r_{i_N(t)}) = \sigma(t) = \frac{1}{N}\sum_{j=1}^{N}(j - i_j(t))^2, \quad (1)$$

which measures the average squared difference of the current index of random values from the correct index. This measure is minimized when the sorting is perfect, when it takes the value of zero.

The value of this measure is shown in Figure 2. Note that the figure indicates that less than 20 cycles are sufficient to reduce the average error to 1% of the network size, which means that nodes are $N/100$ positions away from their correct position on average, independently of network size. With $c = 80$, 40 cycles are enough to reach 0.1% precision.

4.2. Analogy with Average Calculation

In the average calculation problem, all nodes have a numeric value, and the goal is to calculate the average of these values. The protocol presented in [4] has a striking similarity with our sorting protocol. In that case, in each cycle each node picks a random peer using the peer sampling service (for example, NEWSCAST), and performs an averaging step with the selected peer, during which both nodes replace their value with the average of their two old values. The performance measure for the averaging is defined as

$$\sigma_{avg}(t) = \frac{1}{N}\sum_{j=1}^{N}(w - w_j(t))^2. \quad (2)$$

where $w_j(t)$ is the value at node j at time t, and w is the average of the values.

The local update step is analogous to the swapping of values in the present sorting protocol. The difference is that in the case of sorting, swapping can be performed only with suitable nodes, while in the case of averaging there is no such restriction. In the case of both protocols, we can talk about a *weight conserving* property. In the case of averaging, the local averaging step keeps the global average identical while reducing σ_{avg}. In the case of sorting, it can be seen that

$$\frac{1}{N}\sum_{j=1}^{N} j - i_j(t) = \frac{1}{N}\sum_{j=1}^{N} j - \frac{1}{N}\sum_{j=1}^{N} i_j(t) = 0 \qquad (3)$$

for all permutations. Just like in the case of averaging, a swapping step does not change the value above (it remains zero), while—as it can also be easily proven—it always reduces σ.

4.3. Exponential Convergence

In the case of averaging, it has been proven that σ_{avg} decreases exponentially fast, as a function of the number of local exchanges performed. In particular, it was proven that

$$E(\sigma_{avg}(t+\delta)) \approx E(2^{-\phi})E(\sigma_{avg}(t)) \qquad (4)$$

where random variable ϕ counts the number of local averaging steps a random node in the network participates in during a time interval of length δ.

Motivated by the analogy described above, we propose a similar simplified deterministic model for the sorting protocol:

$$\sigma(t+\delta) = k^{-\phi}\sigma(t), \qquad (5)$$

where $k > 1$ is a constant and ϕ is the number of swaps one node participated in on average. As can be seen in Figure 3, the qualitative prediction of exponential behavior is very accurate. We emphasize that this is an empirical (and not an analytical) model that is otherwise motivated by analytical models of analogous protocols.

4.4. The Number of Successful Swaps

It is remarkable that the prediction of the convergence speed as described above depends only on the number of swaps, no matter how fast the number of successful swaps are performed. This allows us to separate the analysis of the protocol into two independent parts: the speed at which successful swaps are performed and convergence as a function of the number of swaps.

Peers to swap values with are found via unbiased random sampling from the entire network. First of all, the probability that in a certain point in time t a random sample from the entire network is a suitable peer is proportional to the error $\sqrt{\sigma(t)}$. Using this observation,

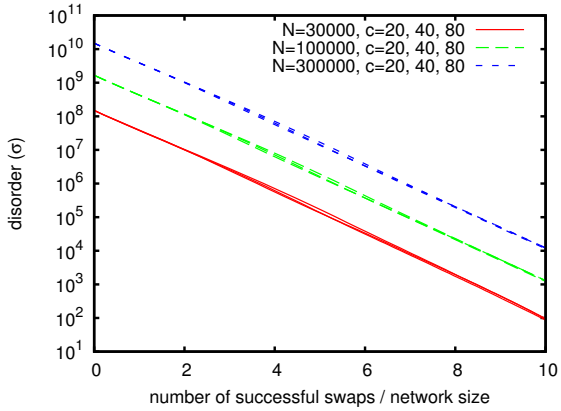

Figure 3. The exponential decrease of the disorder as a function of number of successful swaps (normalized by the network size), for different values of parameter c (view size) and network sizes. Lines that belong to the same network size fully overlap.

we derive a model to predict the number of successful swaps. Let $a(t)$ denote the number of successful swaps at time t. We can define a recursive equation as follows:

$$a(t+1) = a(t)\frac{\sqrt{\sigma(t+1)}}{\sqrt{\sigma(t)}} = a(t)(k^{\frac{1}{2}})^{-a(t)} \qquad (6)$$

where we applied (5).

It can be easily verified, that for large t, the solution is $a(t) \sim t^{-1}$. To see this, let $\log x$ denote the logarithm of base $\sqrt{k}$. Then,

$$\log a(t+1) = \log a(t) - a(t) = \log a(1) - \sum_{i=1}^{t} a(t). \qquad (7)$$

After substituting $a(t) = bt^{-1}$, the equation remains a good approximation if t is large:

$$\log(t+1) \approx C_2 + b\sum_{i=1}^{t} i^{-1} \qquad (8)$$

where C is a constant. The right hand side of the equation is a good approximation of the left hand side for a suitable constant b, due to the fact that $d\log t/dt \sim t^{-1}$.

As shown in Figure 4, the number of swaps depends on the view size parameter c, but in all cases it has a power-law tail, with exactly the predicted coefficient: x^{-1}. In the first cycles however, the number of exchanges remain approximately constant. This is due to the fact that the algorithm uses $c > 1$ candidates to eventually select a peer. If p is the probability that a random peer is a suitable peer, then in each selection step, the algorithm selects a suitable peer with probability $1-(1-p)^c$. While p is large (that is, while t is

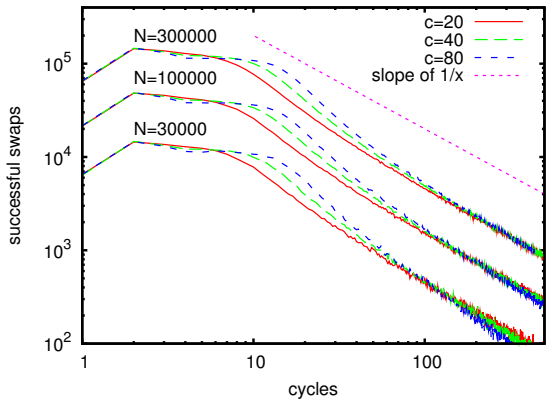

Figure 4. Swaps as a function of cycles. Curves completely overlap when normalized by N.

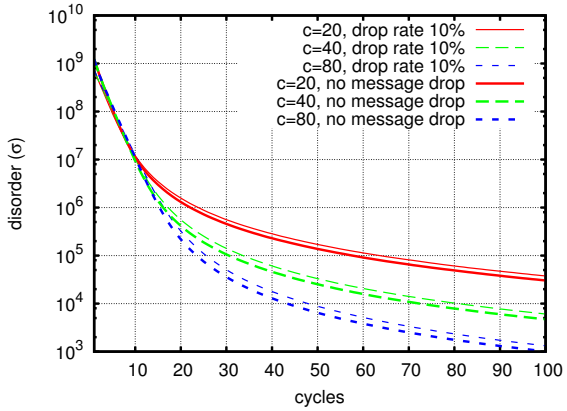

Figure 5. Disorder as a function of cycles, with and without message drop, for $N = 100000$.

small), this probability remains close to one, and as a result σ decreases exponentially fast. However, when p is small, $1-(1-p)^c \approx cp$, that makes our predictions for the tail valid, since in this case the probability of success is proportional to p which is in turn proportional to the actual error.

This analysis tells us that, depending on c, there is an initial phase in which the convergence of the squared error is exponential, followed by a phase when convergence slows down. Most importantly, this result is independent of network size which allows for a scalable and robust setting of parameters.

5. Experimental Analysis

We have performed extensive simulation experiments in order to study the behavior of the protocol in the presence of message drop and node dynamism (churn). The experiments were performed using the PEERSIM simulator [8]. All scenarios were run with three network sizes ($N = 30000$, 100000 and 300000) and three view size settings ($c = 20$, 40 and 80).

5.1. Message Drop

The protocol generates a large number of independent message exchanges (a request and an answer) at all nodes. In the implementation, the messages are assumed to be sent using an unreliable channel, such as UDP, and there is no failure detection mechanism.

If the message carrying the request is dropped, the exchange is dropped as a whole. These cases simply slow down the convergence proportionally to the number of failures, without changing its characteristics.

If the answer is dropped (after the contacted peer has updated its value) then the value originally held by the contacted peer is lost, since the requestor peer will keep holding its original value. In other words, one of the values gets duplicated and the other gets lost. This however has no dramatic effect, as long as there are still a sufficient number of different values, since the distribution of the set of all values is still uniform (since no bias is involved in the message failures). Indeed, as shown in Figure 5, we can only observe a proportional slowdown, under 10% uniform message drop, that can be considered a rather significant drop rate. For other network sizes we obtain identical results. We can conclude that the quality of ordering is highly robust to message drop failures.

However, diversity of the values is important, because the "resolution" of the system (the number and size of the groups it can order) depends on this diversity. The dynamics of the number of unique values is interesting and complex, that we cannot address here due to lack of space. Very briefly, if there is no churn, then there will be fewer and fewer swaps as the system converges to the ordered state, as described in Section 4. Besides, when each value still represented has a small number of copies, it becomes very unlikely that all copies of a specific value are completely removed. Due to these two properties, diversity practically stops decreasing very soon. In addition, in the presence of extreme failure rates, we can add a simple technique to further fight the lack of diversity: whenever a node sees another node in its view that holds the same random value, it replaces its own value with a random one. This technique in effect introduces a very low rate artificial churn, that is dealt with just like real churn (see Section 5.2). We also note that if there is natural churn, then diversity is maintained by the churn itself.

5.2. Churn

To examine the effect of churn, we define an artificial scenario in which a given proportion of the nodes

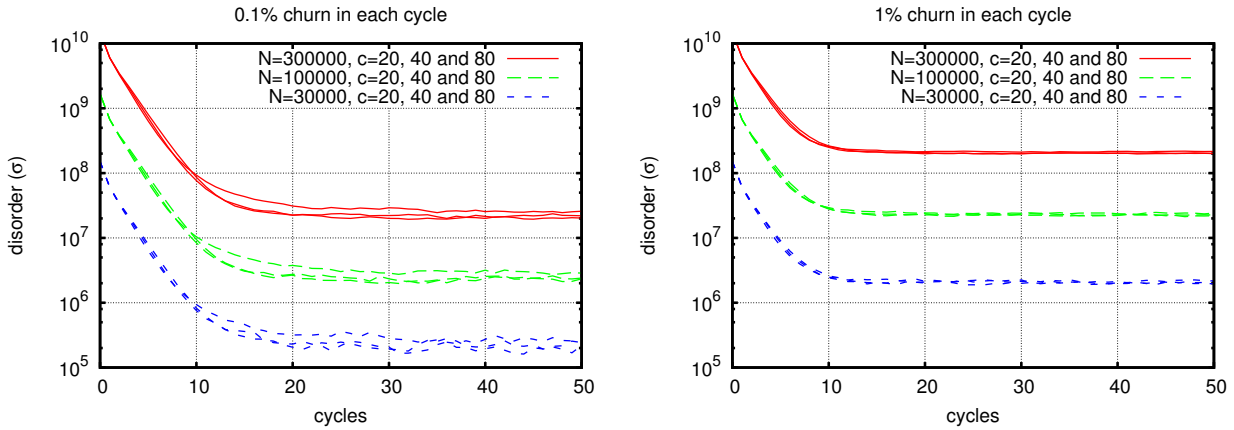

Figure 6. Disorder as a function of cycles, for churn rates 0.1% and 1% per cycle. Curves completely overlap when normalized by N^2.

crash and are subsequently replaced by new nodes in each cycle. This scenario is a *worst case* scenario because the new nodes are assumed to join the system for the first time (their random value r_i is independent of their attribute value x_i) and the crashed nodes are assumed never to join the system again. The view of joining nodes is initialized with descriptors of randomly selected nodes.

Churn rate defines the number of nodes that are replaced by new nodes in each cycle. We consider the churn rates 0.1% and 1%. Since churn is defined in terms of cycles, in order to validate how realistic these settings are, we need to define the cycle length. With the very conservative setting of 10 seconds, which results in a very low load at each node, the trace described in [10] corresponds to 0.2% churn in each cycle. In this light, we consider 1% a comfortable upper bound of realistic churn, given also that the cycle length can easily be decreased as well to deal with even higher levels of churn.

The results of the experiments are shown in Figure 6. The ordering effort of the protocol and the continuously introduced disorder reaches an equilibrium after a few cycles, after which the level of order remains stable. Even with a 1% churn rate in each cycle, the protocol manages to keep the average distance from the correct position by approximately an order of magnitude less than that in a random permutation. Note that in this scenario, during the 50 cycles shown, almost half of the network gets replaced at least once.

We can further improve the performance of the protocol using techniques that take the age (time spent in the network) into account. One technique is called *age bias*; when using this technique, a node, when selecting the neighbor to swap with, chooses the one among the candidates which has the most similar age. This can be easily implemented without extra communication steps, if the node descriptors in the view also contain node age.

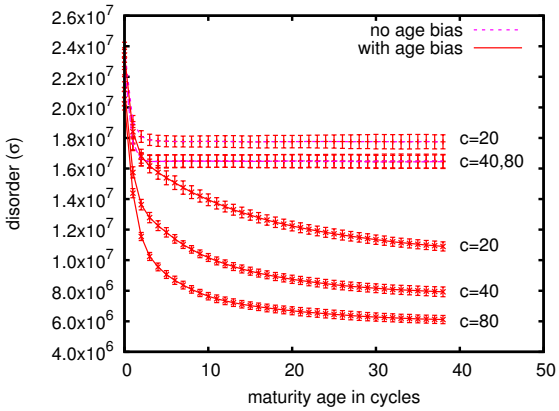

Figure 7. Effects of age-based techniques. The converged value of σ is shown. Network size is 100000, error bars show standard deviation over 50 cycles (cycles 50–100).

As a result, only the younger nodes tend to be disordered, while they can still converge and while the older nodes that have already converged remain protected. Indeed, as shown in Figure 7, we obtain a considerable improvement using the age bias technique, if, in addition to the age bias, we also require a certain maturity (that is, minimal age) to be considered as part of any slice. In other words, the order among the nodes that have a certain minimal age improves significantly.

5.3. An Illustrative Example

To illustrate how well our approach copes well with highly dynamic environments, Figure 8 provides a visualization of three slices that are maintained in a network of size 1000, over 1200 cycles, using age bias and a maturity level of 20 cycle. The slice specification is $(1/3, 1/3, 1/3)$, we have three slices of equal size. The

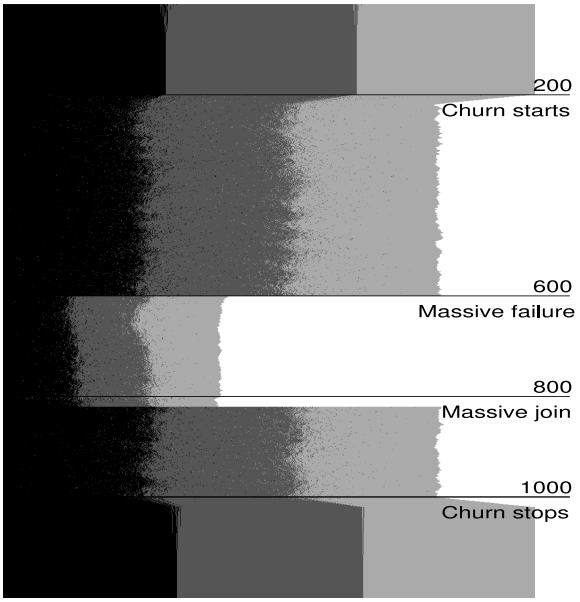

Figure 8. Visualization of groups in extreme failure scenarios.

view size is $c = 20$. After the start of churn the network seems to shrink. This is due to the fact that we consider only mature nodes, that is, those that are older than 20 cycles.. The scenario we applied includes churn (1% in each cycle), removal of a random half of the network and subsequently duplicating network size in one cycle. We observe that the slices remain relatively well defined, especially if we consider that the entire network gets replaced several times during the period shown. We also observe that as soon as the churn stops, the slices get stabilized as well. Note that our goal cannot be to eliminate churn within a slice completely, but only to make sure it is similar to the churn the entire network is experiencing.

Finally, the fault tolerance of the underlying NEWSCAST protocol in similar scenarios has been discussed elsewhere [3], where it was shown to be extremely robust.

6. Conclusions

In this paper, we have described a solution to automatically partition a highly dynamic network according to a given metric as well as to maintain such a partitioning in the presence of churn. In our approach to the ordered slicing problem, each node has to identify which section of the network it belongs to, ordered along an attribute x_i, using only local information. Our solution relies on a robust and scalable gossip-based sorting protocol. We have presented approximative theoretical results based on an analogy with average calculation and demonstrated the robustness of the protocol in simulation experiments.

In this paper we focused only on the identification of the slices, which is in itself a challenging problem. However, to be practically useful, these slices have to be presented to users and applications as groups. We are currently working on this issue. In a nutshell, our solution to this problem is to execute a slice specific NEWSCAST protocol inside each slice, which implements the peer sampling service (providing samples from the slice). Users of a slice will simply be part of the slice and access it through the peer sampling service. Nodes (and users) can join a slice-specific NEWSCAST via a random contact from the slice. Such contacts can be stored (and continuously updated) together with the slice specification, which, as we mentioned previously, can be thought of as a very small database stored at each node and maintained cheaply using anti-entropy gossip.

References

[1] D. P. Anderson. BOINC: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, 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 *Proc. NSDI*, pages 253–266, 2004.

[3] M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In H.-A. Jacobsen, editor, *Middleware 2004*, volume 3231 of *LNCS*, pages 79–98. Springer, 2004.

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

[5] M. Jelasity and M. van Steen. Large-scale newscast computing on the Internet. Technical Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, October 2002.

[6] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. FOCS'03*, pages 482–491. IEEE Computer Society, 2003.

[7] A. Montresor, M. Jelasity, and O. Babaoglu. Decentralized ranking in large-scale overlay networks. Technical Report UBLCS-2004-18, University of Bologna, Dept. Comp. Sci., December 2004.

[8] PeerSim. http://peersim.sourceforge.net/.

[9] J. Sacha, J. Dowling, R. Cunningham, and R. Meier. Using aggregation for adaptive super-peer discovery on the gradient topology. In *Proc. SelfMan 2006*. Springer.

[10] S. Saroiu, P. K. Gummadi, and S. D. Gribble. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia Systems Journal*, 9(2):170–184, August 2003.