Fault Tolerance of the Application Manager in Vigne

Rajib Kumar Nath\textsuperscript{1,2} \\
\textsuperscript{1}IRISA/Paris Project-Team, \textsuperscript{2}University of Tennessee, Knoxville \\
rnath1@cs.utk.edu

1 Introduction

Failure is a common phenomenon in distributed systems. As a system gets larger and complex, the numbers, sources and types of errors increase proportionately. To implement a system that works properly despite failures has been a major concern in distributed system research community for the past few decades. Researchers have been trying to enhance fault tolerance in every sector of computing such as web servers, storage, microprocessor, communication channel, application components etc. but ensuring fault tolerance in a large system like grid has been a very challenging and troublesome job. Grid is a collection of computing resources located in different administrative domain. Because of this large volume, it’s not easy to manage grid. The grid middleware was introduced to fulfill this need. Usually a user submits a job through grid middleware and gets the result back after the computation is over. Throughout the life time of the job, a grid middleware service component manages the job on behalf of the client. In the grid middleware Vigne [39, 32], developed by Paris project team in IRISA, Application Manager (AM) is the service component that supervises any application. It is the critical component of the system. That’s why we want it to keep running despite any failures or reconfiguration that can occur in the grid. The goal of our work is to find solution to increase the availability of AM in Vigne.

This report is organized as follows. Section 2 gives a short description of Vigne and presents the problem informally. Section 3 recognizes critical issues in a fault tolerant system. Then we describe the existing fault tolerance techniques and compared them in section 4 to select a suitable technique for replicating AM. Section 5 and section 6 make the necessary specifications to integrate the selected technique in Vigne. Section 7 addresses crucial issues regarding reconfiguration in highly dynamic environment. Related works are presented in section 8. Finally we present the conclusion and future work in section 9.
2  Context

In this section we give a short description of Vigne, its underlying peer to peer network and some of its service components. We also present our problem of fault tolerance of the AM and address the necessity of replicating other services. At the end, we present our assumption about the system.

2.1 Vigne and it’s Three Basic Principles:

Vigne is a grid middleware that have been developed in the first place to make use of various types of numerous computing resources located in different geographical locations and to serve the scientific computing community. Any system (e.g. cluster, personal computer) can connect to or disconnect from Vigne at any time and thereby used for computing jobs submitted by client. As Vigne possesses no control over any of these resources, failures and node arrivals are very frequent in the system. Any resource can be manually restarted or stopped by the owner, or there may be failures. This dynamism has put Vigne in a position of enhancing fault tolerance of an ongoing computation as well as of its service components.

Because of heterogeneous resource types and dynamic environment, three basic principles have been maintained during the design and implementation phase of Vigne: Single System Image, Self Healing and Self Organization [39].

- Single System Image: Users are provided with a simple abstraction that hides the physical distribution and types of resources.
- Self Healing: In case of node addition, removal or failures, Vigne automatically reconfigures while preserving transparency with users. It also provides transparent and generic fault tolerance for computations.
- Self Organization: The owner of a resource has to maintain the local setting of his resource. Once the resource has been integrated to Vigne, the responsibility of efficient use of the resource is delegated to Vigne.

2.2 Network and Service Components of Vigne:

The underlying network of Vigne is a peer to peer network [2, 29]. The choice of peer to peer network naturally follows from the last two basic principles mentioned in section 2.1. Vigne is implemented on top of Pastry [29, 34] structured overlay [17] with maintenance algorithm of BAMBOO [2]. Any node can join the overlay of Vigne by contacting a node that has already been included in the network of Vigne. Meanwhile the principle of Single System Image has motivated Vigne to include high level distributed services such as application scheduling, persistent data management, volatile data management [33] and low level services such as resource access control and membership [17]. Among all these services, the two most important services are Application Manager and Resource Allocator which are built on a peer to peer infrastructure. They are described in next two subsections.
2.3 Application Manager in Vigne

AM is a dedicated agent that supervises an application throughout its lifetime. Whenever a client desires to submit a job, he or she contacts a node that is already part of the Vigne peer to peer overlay and thereby handovers the job description file. The Application Scheduling Service of the contacted node chooses an Application Manager Key (AM-Key) randomly. Then an AM is created in a node whose id in the p2p overlay is the closest to the AM-Key among all the nodes’ id in Vigne on that particular moment and the client is informed with the AM-Key. In future the client can make any query or modification of the application using this AM-key because any application related message is routed to the right AM in the structured overlay. Since the submission of the job by the client, AM acts on behalf of the client to run the computation successfully despite any change in the peer to peer overlay. All resource necessary for the computation is provided by the AM by contacting the Resource Discoverer. The fault tolerance of the application process is also provided by the AM [32, 17]. The current picture of AM and client interaction in Vigne is shown in Figure 1 (Here AP stands for Application Process).

Unfortunately its way too likely that host node of AM will fail. So it is highly important for Vigne to increase the availability of AM. Otherwise in case of failure of the AM’s host node, the Application will be running without any coordination. It may stop on the halfway waiting for the failed AM’s signal and the client will never get the result or status of the submitted job. As a result allocated resources might not be freed. The goal of our work is to tolerate such failure. With this goal in mind, we have evaluated the existing fault tolerance techniques and chosen the best suitable one. Then we have made specifications and design choices to integrate this feature in Vigne.

2.4 Resource Allocator in Vigne

The Resource Allocator (RA) is the component that searches, selects and allocates most suitable resources for an application according to the job specification given by the client during the job submission phase. The job specification in-
cludes processor architecture, number of processors, operating system, free disk space, scheduler etc. The resource discovery is performed using unstructured peer to peer overlay as some of these job requirement might not be a scalar criteria (e.g. memory, free disk space). Currently there are three protocols in Vigne for resource discovery: (a) Basic Flooding, (b) Random Walk, and (c) Reinforcement Learning.

The placement or activity of the RA in Vigne is pretty analogous to AM. It is application specific and RA resides on the same physical node as AM is. The failure or node arrivals in the Pastry DHT † namespace affect RA as the same way it affects AM. Even if we replicate the AM, we will fail to maintain consistency. In case of a failure of RA’s host node, all the application specific information will be gone. A neighboring node will start acting as RA for the corresponding application and the new RA will have no way to figure out whether the upcoming request is a duplicate request that has already been served or it’s a new request. Hence we need to replicate the RA as well. The same concern applies to Application Monitor. For this study, we assume that RA(Resource Allocator) and Application Monitor remains available all the time. We only focus on the failure of the AM(Application Manager).

2.5 Assumption about the System

The correctness or validity of any distributed algorithm or system lies on some assumption. Our assumptions are as follows:

- We will consider only Fail-stop failures. Byzantine failures [23] are not taken into account as they can be handled using cryptographic algorithm and error check-sum with some majority rules.

- The underlying communication channel is unreliable and non-FIFO.

- We didn’t assume any distribution about the node arrival or failure rate in the peer to peer overlay. Any node at any place in Pastry DHT namespace can fail or join at any moment.

- The other system service components (e.g. Resource Allocator, Application Monitor) remains available all the time.

3 Critical Issues in a Fault Tolerant System

The following issues regarding fault tolerance need to be considered during the design phase of a fault tolerant AM in Vigne:

†DHT stands for Distributed Hash Table and it performs the purpose of a hash function. Someone can store a \( \langle \text{key}, \text{value} \rangle \) pair and lookup the value later if he has the key. The name DHT comes from the property that storage and lookups in the DHT is distributed among multiple nodes. This inherent property has made DHT popular in fault tolerant peer to peer computing.
• **Reliability**: Vigne should avoid single point of failure, otherwise there would be no way to recover. It should also avoid single point of repair for analogous reason.

• **Scalability**: The system should be decentralized so that huge amount of client requests are evenly distributed among available resources. As Vigne should be capable of managing thousands of Applications on numerous grid nodes, any centralized solution will put Vigne’s performance in jeopardy.

• **Transparency**: The property of Single System Image in Vigne should be preserved. Any replication should remain transparent to Application Process and specifically to clients.

• **Consistency**: The system should always compute and deliver consistent result even if there is a failure.

### 4 Existing Fault Tolerance Techniques

In this section, we describe and evaluate existing fault tolerance techniques to choose the best suitable one for replicating AM in Vigne. There are several techniques to achieve a fault tolerant system in literature. The common hierarchy of fault tolerance techniques is show in Figure 2. As we can see there are two major trends: Flat and Hierarchical depending on the presence of hierarchy in the fault tolerant mechanism. The characteristics of different techniques are described below.
4.1 Active/(Cold)Standby Replication with Shared Storage

The active service component stores its current state in a stable storage infrequently while serving clients’ request. An analogous standby entity monitors the health of the original service component through heart beat mechanism. If the standby monitoring component suspects the active component as faulty, it restores the last saved state from the stable storage and takes the role of the active component (Figure 3).

4.1.1 Benefits

- **Low Redundancy**: It needs less computing resources than most of the replication techniques. Besides there is no software redundancy as the standby component doesn’t do any computing other than monitoring the active component.

4.1.2 Drawbacks

- **Single Point of Failure**: It uses a shared storage to store the state of the service. But stable storage may not always be available. Even though it’s available somewherer in the grid, the physical node that is holding the AM may not have any stable storage. It’s true that with the help of RA, we can resolve the problem. But the owner of the stable storage may take it out of grid any time he or she wants. Hence technically, it will turn out as a single point of failure.

- **Loss of State**: As the service has to roll back, it will lose states between the last checkpoint time and the time of failure event.

- **No Seperation of Responsibility, Less Flexibility**: If we are using checkpointing for fault tolerance of AM then it will be efficient if the checkpointing of application process and AM are not decoupled. Hence we will lose some flexibility.
• **Parametric Solution:** The efficiency of the system will depend on the frequency of the checkpointing. If we take the checkpoint more frequently then it will be inefficient and the processes will do more checkpointing than the original job that they have been assigned for, but it will increase availability. If we do checkpointing less frequently then we run the risk of losing massive amount of computation in case there is a failure between last checkpointing and just before the next checkpointing. Hence the checkpointing period is a critical parameter for the system and it will be hard to calibrate such parameter because of the dynamic environment of grid.

• **Low Performance and Implementation Hardship:** The main bottleneck in checkpointing algorithm lies in the time required to write states on the stable storage. To take a checkpoint, all the processes have to agree on some snapshot period according to some global snapshot algorithm (e.g. Chandy-Lamport). This will incorporate some communication overhead and hamper the performance of the applications and networks.

  It will not be easy to decide the checkpointing time. In Vigne, if we think about the application submission period, we will find that there are two different times when we might do the checkpointing: after resource discovery phase is over or after sending back the application key to the client. Each of the choices has its own drawbacks. Whatever we choose here, we have to take care of these cases with logical checking in the related code. On the other hand, if we want the checkpointing in every such case, it will increase the number of checkpoints. As the RA is involved throughout the life time of an application, many such cases will arise. Thus it will lead us to implementation hardship (e.g. changing the existing code) as well as to low performance.

• **Low Availability:** A substantial startup time is required for the standby to take over the failed one and update it’s states. In Vigne, the application processes may have to rollback and restart from the last checkpoint. Hence the service will be unavailable for certain amount of time.

• **Lack of Transparency:** A separate entity is required to detect the failure of the active component. Usually in the Active/ (Cold) Standby system, the standby component stays idle and keeps checking the health of the active component through heart beat mechanism. If the standby entity isn’t replicated itself, it will be a single point of failure. If we want to tolerate n failures at a time, we need n standby components waiting in a chain fashion. First standby will monitor the active component. Second standby will monitor the first standby and so on. But n such failures can occur in such a pattern that the system might be unavailable for a substantial amount of time (e.g. n * MTTR, here MTTR is mean time to recover). In Vigne, if a client comes by this time with a query or job modification request, he will not be answered as there is no active AM. The client might assume that the application manager doesn’t exist.
Otherwise he has to recognize the failure of the active AM and wait for the standby component to take over. This will reduce transparency.

4.2 Active/(warm)Standby Replication:

In this type of replication technique, the active service component updates its state to an analogous standby component infrequently. The standby component monitors the health of the active component periodically through heart-beat mechanism. In case of a failure of the active one, the standby takes over (Figure 4).

4.2.1 Benefits

- **High Performance:** There is no costly write on the stable storage. So it will provide better performance in terms of Latency experienced by client.

4.2.2 Drawbacks

- **Loss of State:** As there can be a failure between the last update and the next update, some state will be lost.

- **Single Point of Repair:** The standby component is the single point of repair if there is only one standby. We might desire to increase the degree of fault tolerance by keeping several standby replicas. If standby replicas act on chain fashion, it faces the same problem of availability and transparency like active/ (cold) standby. Besides all standbys might not be in the same state depending upon failure pattern and update policy. Hence the amount of state loss will increase.

- **Redundancy:** It requires both hardware and software redundancy. Both kind of redundancy come from the monitoring component as we are using extra resource and that resource is doing the same computation as the service component.

![Figure 4: Warm Standby](image)
4.3 Manual Masking:

Manual masking requires human action to detect the failure and replace the failed component with a good one. We can discard this from our pool of fault tolerance techniques because it will compromise the self healing property of Vigne.

4.4 Active/(Hot) Standby Replication with Primary Backup

The component to be replicated is called the primary. Primary is the only one that communicates with outer worlds (client or Application Processes). One or more standby components are maintained. The standby replicas are always kept up-to-date with the current state of the primary replica so that in case of a failure of the primary, a standby replica can take over automatically from the last consistent state without any state loss (Figure 5).

4.4.1 Benefits

- **No State Loss:** As the primary and standby replicas follow same sequence of states, no state is lost.

- **Easy Implementation:** It’s easy to implement from the programmers point of view given that there is a reliable failure detector. If there is none then it’s hard to implement in asynchronous system. This shortcoming can be overcome by primitives provided by View-Synchronous paradigm.

- **Separation of responsibility:** In Vigne, we can easily separate the fault tolerance of Application Process and AM if we use this technique. Both of them can be handled separately with different policies. If both responsibilities are decoupled then the policy of fault tolerance of an application can be specified by the client before the start of the application depending upon some metrics (e.g. importance of the application, priority of the clients, critical characteristics or the available resources of the
system). If checkpointing is selected for the fault tolerance of application then the interval between two checkpoints can be varied dynamically depending upon the failure rates in DHT namespace. On the other hand if the Active/(hot)Standby replication mechanism is chosen for Application Processes fault tolerance, then the replication policy and degree can be adjusted runtime. This will enhance the self healing property of Vigne.

4.4.2 Drawbacks

- **Low Scalability**: The primary replica will always be a bottleneck as he is the only one to receive or send messages from or to outer worlds and to take care of the state commit among the replicas. Hence a distant client relative to primary replica will experience a high response time.

- **Lack of Transparency**: If a standby replica fail-stops then it remains transparent to the client. But the failure of the primary replica may not remain transparent to the client anymore. In Vigne, the Application Process or client has to resend some of the messages in case of primary AM’s failure.

- **Low Availability, High Latency**: In case of failure of the primary, it takes a substantial amount of time for a standby to take over. So for a while the service remains unavailable. In Vigne, this will lead to increased latency experienced by the client.

  More precisely it depends on group membership for keeping consistent state among the replicas. When the primary replica fails, the group needs to be adjusted and a group communication is started to select the new primary dynamically. This dynamic group management introduces substantial overhead.

4.5 Symmetric Active/Active Replication

In case of symmetric active/active replication, two or more analogous service components send or receive messages to or from clients or Application Processes by maintaining same sequence of global states. They use advanced state commit protocol to keep the state consistent in every replicas. A common architecture of symmetric active/active replication is given in Figure 6. This figure is taken and modified from Engelmann et al.

4.5.1 Benefits

- **No Single Point of Failure or Repair**: There is no single point of failure which is the most desirable property of a fault tolerant system. As long as majority of replicas are non faulty, the system will go on without any interruption and make any necessary reconfiguration. Hence there will be no single point of repair.
• **No State Loss:** As all the replicas maintain same sequence of states from the very beginning, a failure will not cause any state loss.

• **High Availability:** No startup time is required in case of failure. If there are total \( n \) replicas, then the system will be running in a consistent state without any interruption as long as some replica is non-faulty.

• **Transparency:** No separate entity is required to startup a new replica (AM) in case of a failure. This will enhance transparency.

• **Scalability:** As every replica is capable of handling requests, the system will be scalable. A client will receive replies from its closest replica.

• **Reconfigurability and Flexibility:** As replicas are using group communication, a mechanism can be incorporated to maintain the degree of fault tolerance. For example, in case of a failure, a new replica will be created starting from the current state of the replicas. Hence there will be more automation and more transparency. The degree of fault tolerance can be adjusted dynamically depending upon the current churn rate in system and availability of resources.

  If vulnerability is suspected in some replica, then it’s easier to replace it (one at a time) with a good one while keeping the system running. This scenario arises when the owner of a resource (e.g., cluster) wants to make some upgrade in his system. Sometimes replicas (or part of the system e.g., disk) need to be replaced depending upon the data (e.g., Mean Time to Failure) given from the manufacture.

• **No Idle Resources:** It doesn’t waste resources by keeping them idle.

• **Separation of responsibility:** The same benefit is available here as we discussed in Section 4.4.1 for hot standby replication.
4.5.2 Drawbacks

- **Doesn’t allow Non-determinism**: With the Active/Active replication we have to make sure that there is no source of non-determinism. Some sources of non-determinisms are multithreading, local timers, local random number generator etc. If we want to enhance performance through multithreading (like getting benefits from multi-core architecture, overlapping I/O and CPU activities) then we need to ensure same thread interleaving in all the replicas. But it will damage the scalability and performance of the replicated system.

- **Low Performance and High Latency**: It is based upon atomic broadcast which is costly and thus leads to high latency. Hence it’s going to reduce performance of the system. The performance will depend on the geographical location of the replicas as well as on the network links. If they reside in adjacent or closer geographical site then the implemented atomic broadcast will be fast. Whereas if they reside in distant geographical locations, then atomic broadcast black-box will be a bottleneck for the whole system.

4.6 Asymmetric Active-Active Replication

This kind of replication technique is used when there needs to replicate two or more services. In Figure 7, we can see a standard architecture of Asymmetric Active/Active replication. Two head nodes are running two different services.
The standby node in the middle is the back up for both head nodes. In case of failure of either of the head nodes, the standby node assume the IP address and host name of the failed one and keep continuing normal operations without any interruption. The architecture is designed in such a way that the failure of any head node remains transparent to the client and there is no state loss. More details can be found in [24]. This type of replication technique is widely used in telecommunication field. As we are only replicating one service and there is no coordination between the active components, we can easily discard it from consideration. Besides the special hardware requirement will put restriction on the AM’s host node. As the AM is built on top of structured overlay, it won’t be always possible to meet such constraint.

### 4.7 Hierarchical Based Fault Tolerance

One example of this kind of fault tolerance technique is developed in [40]. We are giving a brief description of that system to give the user a quick insight of such techniques. Any entity in that system can take one of the five roles e.g. Executor, Manager, Intermediate, King, Prince. The hierarchy of roles is shown in Figure 8. By hierarchy a derived role will inherit behavior of it’s parent (e.g. everyone will inherit the behavior of Basic, Manager’s role will be inherited by Prince, King and Intermediate). The role of the Executor and Manager is to
run Application Process and grid system services respectively. Intermediate is the standby for the Manager and usually it acts as an Executor. The King handles event like failure and rejoin event of Managers and thus enhance the self-recovery protocol. Prince is the standby for King and it keeps monitoring the King throughout the role period. Each of the roles has different ranking. In Figure 9, we can see that the ranks increases from Executor to King by arrow. With the diversity in roles and ranking, they have developed a grid operating system with self recovering and rejoining protocol. More details are available in [40].

Unfortunately it doesn’t fit well with the flat structure of Vigne and in the context of AM.

4.8 Choosing a Replication Technique for Application Manager in Vigne

Table 1 summarizes the pros and cons of existing fault tolerance techniques in the context of the AM in Vigne. We have already ruled out Asymmetric Active/Active replication and Hierarchical replication techniques because of their incompatibility with the structure of Vigne. As we mentioned in 4.3, we have also discarded Manual Masking. For other replication techniques, we made a qualitative comparison mainly based on the performance metrics described in section 3. Few other metrics (Flexibility and Latency) have been used. In Table 1, the good side of a technique is emphasized with + and bad side is emphasized with -. From table 1 we can see that Symmetric Active/Active replication is the most desirable one for AM’s fault tolerance in terms of availability, transparency, latency, reconfigurability, flexibility and scalability. The latency and scalability metric are the areas where Symmetric Active/Active has outperformed Active/(Hot)Standby replication technique.
<table>
<thead>
<tr>
<th>Replication Technique</th>
<th>No Single Point of Failure</th>
<th>Availability</th>
<th>Transparency</th>
<th>Flexibility</th>
<th>Latency</th>
<th>Scalability</th>
<th>Loss of State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric Active/Active Replication</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Active/(Cold)Standby Replication with Shared Storage</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Active/(Hot)Standby Replication with Primary Replica</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Active/(warm)Standby Replication</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1: Qualitative Comparisons of Replication Techniques
5 Symmetric Active/Active Replication in Vigne

In this section, we describe Symmetric Active/Active replication formally and analyze the feasibility of its implementation on top of a peer to peer framework. Then we address the kind of group communication infrastructure we need for replicating AM in Vigne.

5.1 Basic Principle

A standard definition of Symmetric Active/Active replication can be found in Engelmann et al [12]. As we have described before, Symmetric Active/Active replication ensures the availability by maintaining more than one analogous active service components throughout the life time of the service. Like most of the standby solutions, it doesn’t waste resource while keeping them idle. All the replicas are capable of serving client requests while running in virtual synchrony without any need of failover. So there is no state loss or any service interruption. The correctness of symmetric active/active replication lies in the following properties.

1. Each replica runs in a non faulty node.
2. All replicas start from an identical initial state.
3. All replica process same sequence of commands deterministically.

With these three properties, they produce the same sequence of output by maintaining same sequence of states. Replication is performed by total ordering [20] all state change messages and reliably delivering them to the replica set. A process group communication is used to ensure total message order and reliable delivery. The group communication also manages the replica set in case of failure. Furthermore, consistent output produced by all active replicas, may be routed through the group communication layer if the receiver of the output (e.g. client, Application Process, Resource Allocator, Application Monitor in case of Vigne) doesn’t have the ability to recognize duplicate message. As mentioned above in Section 4.5.1, it has the flexibility to enhance degree of replication by modifying the replica set dynamically in case of high failure rate.

5.2 Active/Active Replication on top of DHT

In this subsection we describe the underlying peer to peer network of Vigne. We also present our choice about physical nodes where all the replicas of AM will reside. Then we describe inherent problem of reconfiguration in context of the dynamic environment of Vigne.
5.2.1 Pastry

Pastry is a self-organizing overlay network of nodes where each node is assigned a 128-bit unique node identifier (nodeid). The nodeid of a new joining node is assigned such a way that the set of all existing nodeid are uniformly distributed. When a message inserted with a destination key in the overlay, the message is routed to the node whose nodeid is numerically closest to the destination key, among all existing nodes in the overlay on that particular moment. If there are \( N \) existing nodes in the current overlay, Pastry can route a message to any destination in less than \( \log_2 N \) steps on average (\( b \) is a configuration parameter with typical value 4). For efficient routing, each Pastry node maintains a routing table and a leaf set. Leaf set contains IP addresses of the set of nodes with \( l/2 \) numerically closest larger nodeid and \( l/2 \) nodes with numerically closest smaller nodeid relative to the host node’s nodeid (\( l \) is the maximum size of the leaf set with typical value 16). Pastry guarantees eventual delivery of a message unless \( l/2 \) or more nodes with adjacent nodeid fail simultaneously. Pastry also uses locality to minimize the distance traveled by a message. Its decentralized, scalable, self-organizing and efficient. For these four properties, Vigne has been built on top of Pastry. For reconfiguration of the leaf set and routing in case of failure or node arrivals, Vigne uses BAMBOO algorithm.

In Pastry peer to peer structured overlay, the leaf set forms a ring as shown in Figure 10. Application manager key (AM-Key) is randomly created after the job is submitted by the client. The AM is created in a node whose nodeid is numerically closest to the AM-Key among all the existing nodes in the overlay on that moment and the AM-Key is sent back to the waiting client. Any messages associated with that application is routed through the structured peer to peer overlay of Pastry using this AM-Key.
5.2.2 Location of the Replicas

In case of replicating AM, we need to choose a set of $2n+1$ nodes where replicas will reside. Here $n$ is the degree of fault tolerance. There are two choices: (a) $2n+1$ nodes that are physically closest, (b) nodes that are numerically closest to the AM-key in the logical ring (Figure 10). The choice (a) is good in terms of performance as all the replicas will reside closely in geographical point of view and the underlying group communication will be considerably fast. But then we will run the risk of putting all the replicas in the same cluster and thus losing all the AM’s in case of failure in the whole cluster. With choice (b), the probability of putting all the replicas in the same cluster will be reduced because of the uniform distribution of keys in DHT namespace. So the natural choice of replica set will be $2n+1$ numerically closest nodes in the current overlay relative to AM-Key. We are going to choose $2n+1$ replicas from the leaf set in Pastry layer. As the replica set will always be a subset of leaf set, any failure information in the leaf set can be used to detect the failure in the replica set.

Another way of selecting replica set is as follows. The replica set of a AM will contain the node whose nodeid is the closest to the AM-Key. Immediate $n$ left neighbors and $n$ right neighbors of the previously selected node will be in the replica set as shown in Figure 11. We are going to follow this selection mechanism.

5.2.3 Study of Reconfiguration Cases

Throughout this subsection, the degree of fault tolerance is taken as one. So we need three replicas to tolerate one failure. The set of replica is named as replica chain and the set is marked by a shaded region in the corresponding figures.

---

† Later in Section 6, we will see that we have selected a consensus based protocol for implementing Symmetric Active/Active replication in Vigne. In asynchronous distributed system with failure, consensus requires minimum $2n+1$ nodes to tolerate $n$ failures.

‡ We could have taken $2n+1$ closest nodes as replica set. There isn’t any relative advantage or drawback between this two ways.
The replica whose key is the closest to the AM-Key is called the central replica. All other replicas will be referred as wing replica.

We have already mentioned that a process group communication among the replica set is used to ensure total message order and reliable delivery. We will give the necessary specification of it in section 5.3. For the time being let’s assume that the regarded group communication is performed using Virtual Replica set †.

Let’s assume that there are two basic types of application related messages in the system with Virtual Replica set: Application Message and Shadow Application Message. An Application Message is delivered to the central replica using the routing algorithm of Pastry structured overlay. Upon receiving a Application message, the central replica creates a analogous Shadow Application Message and sends the message to its closest left and right neighbors in the leaf set using IPaddress. ‡ The central replica also consumes the Application Message itself. The receiving of a Shadow Application Message doesn’t create anymore message §. Upon receiving a Shadow Application Message, a replica handles it as an Application Message and doesn’t forward it to any other nodes anymore.

It may happen that there is one failure or one node arrival inside the replica set DHT namespace. In Figure12, four distinguished cases are presented: (a) a new node arrival whose nodeid is closer to the AM-Key than former central replica’s nodeid, (b) a new node arrival whose nodeid is no more closer to the AM-Key than nodeid of the former central replica, (c) failure of the central replica, (d) failure of a wing replica. In all the four cases some replica receives the application messages given that there is no failure during the forwarding period.

There may be two failures¶ or two node arrivals inside the replica set namespace in DHT. In Figure13, four distinguished cases are presented: (a) two new nodes arrival; one of which id is numerically closer to the AM-Key than former central replica’s nodeid, (b) two new nodes arrival where no one’s id is closer to the AM-Key than nodeid of former central replica, (c) failure of the central replica and a wing replica, (d) failure of two wing replicas. In all the four cases some replica receives the application messages given that there is no failure during the forwarding period.

Three new nodes may arrive at the same time inside the replica set namespace in DHT. In Figure 14, four distinguished cases are presented: (a) three new nodes should be the new Virtual Replica set, (b) the former central replica and two new nodes should be the new Virtual Replica set, (c) a former wing

---

† A few researcher has addressed Virtual Replica set in literature. It simply means that no information about the replica set corresponding to a AM is maintained in anywhere. At a given moment, k nodes closest to the AM-Key forms the Virtual Replica set. Here k is the cardinality of the replica set.

‡ As no physical replica set is maintained, the wing replicas should be the immediate right and left neighbor of the central replica in it’s leafset.

§ Note that this may lead to message duplication in the overlay.

¶ A set of three replicas can’t tolerate two failures in active/active replication. We addressed the case of two failures to represent the problem clearly.
Figure 12: Single Failure or Single Node arrival in the Replica Set namespace

Figure 13: Two Failures or Two Node arrivals in the Replica Set namespace
replica and two new nodes should be the new Virtual Replica set. In cases (b), (c) and (d), some replica receives the application messages given that there is no failure during the forwarding period. But in case (a), no replica will receive the application message or clients requests even though no replica failed. So we have to deal with both failures and node arrivals. Situation like (a) will be very frequent in a dynamic environment like Vigne. Usually this will be periodic events in Vigne e.g. during the morning or midnight. Situation like (a) has cancelled the idea of Virtual Replica set and motivated us to keep a physical replica set for replicating AM on top of structured overlay. It has also necessitates the idea of keeping a framework for reconfiguration purpose. In summary, although the primary purpose of keeping replicas was to tolerate failures, we have to deal with node arrivals in DHT namespace.

5.3 Total Order Broadcast

As we need total order and reliable delivery of messages, let’s investigate which kind of total order broadcast we need for Symmetric Active/Active replication. In [1], total order broadcast is specified on four inherent properties: Validity, Integrity, Agreement and Order. Informally they have been defined as follows:

- **Validity**: A message sent by a correct process is eventually delivered at least by correct processes.
• **Integrity**: For any message m, it will be delivered only once by a process if it was previously sent by some process.

• **Agreement**: At least non-faulty processes deliver same set of messages.

• **Order**: At least all non-faulty processes delivers the same set of messages it receives in same order.

Moreover the dimension of uniformity has given total order specification a new flavor. As we find in [1]:

• **Uniformity**: A uniform property imposes some restriction on the histories of (at least) correct processes on the basis of some events occurred in the histories of some processes.

• **Non-uniformity**: A non-uniform property imposes some restriction on the histories of correct processes on the basis of some events in the histories of some correct processes.

As it is not possible to compel a failed process eventually delivering a message that he has sent, we can ignore Uniform Validity. And if we are in a crash-failure environment, the uniform version of Integrity can be enforced without any additional overhead. So the other versions of Integrity is discarded.

On the other hand depending on the message delivered by a faulty process and strength of delivered message order, there can be four different kind of orders: Strong Uniform Total Order, Weak Uniform Total Order, Strong Non-uniform Total Order and Weak Non-uniform Total Order. So with two choices for Agreement and four choices for order, there can be eight different kind of total order broadcast algorithm. For our active/active replication purpose we have chosen: Uniform Agreement and Strong Uniform Total Order. Because with this specification, all correct processes deliver same sequence of message and all faulty processes deliver a prefix of the sequence of message already delivered by some correct process. So the replicas will maintain same sequence of state and a wrong message from a faulty replica will not compromise the consistency of the system. In literature this kind of algorithm is known as Atomic Broadcast. The selected total order specification is described below as given in [1]:

• **Non-uniform Validity (NUV)**: If a correct process total order broadcasts a message m, then it eventually total order delivers m.

• **Uniform Integrity (UI)**: For any message m, every process p total order delivers m at most once and only if m was previously total order broadcasted by some process.

• **Uniform Agreement (UA)**: If a process total order delivers a message m, then all correct processes eventually total order delivers m.

• **Strong Uniform Total Order (SUTO)**: If some process total order delivers some message m₁ before m₂, then a process total order delivers m₂ only after it has total order delivered m₁.
6 Specifications of the Symmetric Active/Active Replication Architecture

In the previous section, we explained why we have chosen Atomic Broadcast for Symmetric Active/Active replication of AM in Vigne. The purpose of this section is to make the necessary specification to implement Atomic Broadcast in our context. Side by side, we will describe the complete architecture stack needed for Symmetric Active/Active replication and make the corresponding design choice to achieve high performance.

6.1 Atomic Broadcast

There are many different kinds of Atomic Broadcast algorithms in literature that provide the total order property mentioned in section 5.3. We are intending to choose the best available one. From the optimization point of view, there are three classes of atomic broadcast algorithms:

- Algorithm that preserves the states consistent among all the replicas. Here the clients are provided consistent service [30].
- Algorithm that let internal inconsistency arises in server side and keeps the client ignorant about the inconsistency. So there will be no external consistency [19].
- Algorithms that allows both types of inconsistency: internal and external [3, 13]. But later recover the consistency by undoing some message delivery.

We are not allowing any kind of inconsistency in AM replica set or in client side. Because some actions will be hard (probably impossible) to undo. If we allow inconsistency in AM side, to undo it we might have to make lots of modification in AM as well as in other parts of the system (e.g. Resource Allocator, Application Monitor). Besides as the client will be acting as one shot, it will not be possible to undo some actions in client side. The client might have quitted by this time after receiving his result.

There are also different flavors of Atomic Broadcast algorithm from other dimensions. A complete survey of Atomic Broadcast algorithm can be found in [8]. They have made a classification depending upon the mechanism used for ordering messages. We investigated three types of algorithms: Fixed Sequencer Based, Token Based and Destination Agreement on message set.

- **Fixed Sequencer**: A process is elected as sequencer and is responsible for ordering messages. As long as there is no failure, there is only a single sequencer in the system and no other process tries to order messages. In case of the failure of the sequencer, the algorithm is moved to a failure mode where a new sequencer is elected and some messages might be undone depending upon the algorithm. The failure is handled through certain group communication mechanisms [3].
• **Token Based:** This is pretty much similar to fixed sequencer based algorithm except the role of the sequencer isn’t dedicated to a single process. Rather it rotates between processes even if there isn’t any failure. The privilege to order a set of message is represented by a token. A process can order the broadcasted messages if he is the current token holder. The failure of the token holder is handled through certain group communication primitives. The reason behind development such algorithm is to distribute the load among processes [9].

• **Destination Agreement on Message set:** The delivery order is formed by an agreement between destination of the messages. It transforms total order broadcast problem into a sequence of consensus problem. Each instance of consensus decides on the sequence of messages to be delivered [7].

From another dimension, the necessity of physical clock puts all Atomic Broadcast algorithms into two categories: Time Free and Time Based. Most of the atomic broadcast algorithms fall into the category of time free algorithm.

Among all available options for Atomic Broadcast algorithm, we have decided to go with Destination Agreement class (Agreement on message set) mainly because it provides the total order that is identical to our choice (NUV, UI, UA and SUTO). It has sufficient modularity and provides substantial flexibility to replicate an existing system. Here a suspicion of a process in the failure detector layer doesn’t necessarily compel process exclusion. Rather suspicion of processes is used to keep the consensus layer live. It is time free and doesn’t allow any inconsistency in client side or replica side.

The fixed sequencer algorithms have promising performance in the absence of failure e.g. Optimistic Atomic Broadcast [19]. But in case of failure, they perform worse. So we discarded Fixed Sequencer Based Algorithms.

According to many authors, token based algorithms are considered good in terms of throughput. Unfortunately most of the token based algorithms except Ekwall et al [9] are on top of group membership layer. The reason behind avoiding such algorithm is explained in Sections 6.6 and 6.7. Ekwall et al [9] has analogous performance with agreement based Mostefaoui-Raynal [27] and Chandra-Toueg [7] in terms of Latency and Throughput. In [11], it is confirmed that the performance ranking of Ekwall et al, Mostefaoui-Raynal and Chandra-Toueg remains same in wide area network. As Vigne is a large grid, the choice between these three doesn’t matter anymore.

In Summary, we have decided to choose an Atomic Broadcast algorithm that will not allow any kind of inconsistency in AM or in Client or in other part of the system (e.g. Resource Allocator, Application Monitor). Meanwhile the message order of the algorithm should be based on destination agreement on message set to be delivered. Such algorithms are available in [37, 36]. To implement this kind of Atomic Broadcast algorithm we will need a Consensus layer which raises the need of a Failure Detector of its own kind. Hence we need three layers so far: Atomic Broadcast, Consensus and Failure Detector. The inherent property of Atomic Broadcast necessitates the need for a reliable channel.
The replica set for a particular application manager may not be fixed throughout the life time of corresponding application. Because of the failures and node arrivals in Vigne DHT name space, we are compelled to keep a dynamic replica set which will vary in members and size. So we need a group membership layer that will maintain the replica set. As the membership view should be coherent in every replica, all membership change operations need to be totally ordered. So this layer should be on top of the Atomic Broadcast Layer. And to monitor the group and to do the necessary reconfiguration, we need a Monitoring Layer. So the standard stack of Symmetric Active-Active replication will be look like as in figure 15. This Atomoic Broadcast stack was first introduce in Mena et al [26]. The figure has been taken and modified from FORTIKA[15]. A brief description of each of the layers is the context of the rest of the section.

Whenever an AM receives an application related message from the Application Process or client, he sends it to Atomic Broadcast layer so that message is total order delivered in all the replicas. Besides whenever there is a group membership change operation initiated by the Monitoring Layer (Section 6.7), a message is created by the group membership layer (Section 6.6) and sent to
atomic broadcast layer. The Atomic Broadcast layer provides necessary interfaces for these two tasks. When the underlying consensus layer decides a set of message to be atomically delivered, he sends membership change message and application related message to the Group Membership layer and Application Manager respectively. In case of the presence of a replica remove operation in the decision message set of the consensus, he updates the reliable channel about the event so that the sending queue related to that failed peer is cleared.

6.2 Unreliable Channel

The unreliable channel is any physical communication channel where the message transmission isn’t guaranteed or ordered. All messages will be sent through unreliable channel whether it needs a reliable delivery or not.

The underlying Pastry communication framework will be used as unreliable channel.

6.3 Reliable Channel

A channel between P and Q is reliable if it doesn’t create or duplicate messages and every messages sent by P to Q is eventually received by Q given that Q is a correct process.

In real system we can have a fair lossy channel between P and Q where no message is duplicated or created, message can be lost but, if P sends an infinite number of messages to Q and Q executes receive actions infinitely often, then it receives an infinite number of messages from P. Usually the reliable channel is built upon a fairy lossy link through retransmission and acknowledgement mechanisms. A detail description of solving this quiescence problem can be found in [31].

Usually the reliable channel is implemented with TCP [10] send, receive and timeouts. All the upper layers in the stack uses reliable channel to exchange messages with corresponding peers in other replicas. Besides buffer overflow or empty buffer in sending or receiving queue can be used to inform the upper layers (e.g. Monitoring Layer) about potential suspected replicas. This information will be helpful for the Monitoring Layer to issue or refine reconfiguration commands.

6.4 Consensus

The consensus problem is the most basic building block of many agreement problems in distributed fault tolerant application such as atomic broadcast, group membership, atomic commit etc. It is defined by the following three properties:

1. **Termination**: Eventually every correct process decides on some value.

2. **Validity**: If a process decides a value v, then v was proposed by some process.
3. **Agreement:** No two correct process decide differently.

The FLP result showed the impossibility of Consensus in asynchronous distributed system in presence of a single system failure [14]. Later Chandra and Toueg introduced the idea of unreliable failure detector to solve consensus in a system in presence of failure [7]. Hence we need a failure detector to solve the consensus problem in Vigne.

Different Consensus algorithm has been proposed in literature. To mention a few: Chandra-Toueg ◦S [7], Mostefaoui-Raynal ◦S [27] and Lamport’s Paxos algorithm [22, 21]. Paxos requires an Ω failure detector. The description of different kind of failure detectors are available in next subsection.

We have decided to use Paxos because it is safe even if there is failure. The liveness of Paxos can be enhanced through randomization. It can also be implemented without stable storage and performs well in dynamic environment. It can work both with dynamic or static group. It provides concurrency i.e. separate instance of consensus can run independently at the same time. Finally implementing consensus in asynchronous system is a cumbersome job and Paxos is well documented.

Whenever the Atomic Broadcast layer receives some message to atomically deliver to all the replicas in current view, he invokes propose method of Consensus layer. The Consensus layer starts a new instance of consensus if he is the current leader of the replica set and decides the set of messages to deliver in current round and informs all the replicas (including himself) in current view about the decision. Upon receiving the decision set of messages, the Consensus layer delegates the responsibility of handling the messages to Atomic Broadcast layer. Besides the Consensus layer tells the failure detector layer to periodically monitor the leader so that in case of failure of the leader the consensus doesn’t stall forever. All the messages related to the consensus algorithm are transmitted through reliable channel.

### 6.5 Failure Detector

Informally the failure detector is a component in the system that informs the upper layer about the faulty nodes in the underlying network. Depending upon the service it provides (quality of information), there are several types of failure detectors [35]:

1. ◦S: It is defined by two properties: eventually every process that crashes is permanently suspected by every correct process (strong completeness) and there is a time after which some correct process is never suspected by any correct process (eventual weak accuracy).

2. ◦P: It is defined by two properties: eventually every process that crashes is permanently suspected by every correct process (strong completeness) and there is a time after which correct processes are not suspected by any correct process (eventual strong accuracy).
3. Ω: It outputs a single correct process and there is a timer after which all the correct process always trusts the same correct process (eventual leader property).

A good description of failure detectors for asynchronous system is available in [31].

The Failure Detector in Figure 15 is Consensus layer specific. Depending upon the selected consensus algorithm, the choice of failure detector will change. All three types of failure detectors described in 6.5 provides enough synchrony to solve consensus. However it has been proved that δS and Ω are the weakest failure detector to solve consensus in distributed system in presence of failure [6].

As we have already chosen Paxos in consensus layer, the failure detector we need is Ω. The failure detector in the stack is built upon unreliable channel and its purpose is to monitor the leader of the current replica set. In case of a failure of the leader, it will inform the consensus layer about the failure so that the host node can start the current (unfinished) instance of consensus immediately.

6.6 Group Membership

The implementation of AM on top of structured peer to peer overlay and the dynamic environment of Vigne have leaded us to a dynamic group management. Regarding group communication, we need to maintain a view of the groups and provide necessary interface to other layers of the stack (e.g. Monitoring Layer, Consensus Layer, and Atomic Broadcast Layer) so that they can learn about current members of the group. Group Membership layer also provides interface for member addition and removal.

In many implementation of active replication, group membership layer is used for fault detection. But in our case group management and fault detection are completely decoupled. This will add additional flexibility to the implementation and enhance the performance of the system. Because suspicion of a member in the failure detector will not necessarily lead to a membership change and costly state transfer. Thanks to Mena-Schiper for coming up with this valuable idea [26].

Group Membership is located on top of Atomic Broadcast layer to maintain consistent view among all the replicas in case of failure or reconfiguration. Any membership change is atomically broadcasted like application messages in the current replica set. The member addition and removal operations are not initiated by the group membership layer itself. Another layer called Monitoring layer initiates a membership change through the interface provided by the group membership layer. We will see later that Monitoring Layer performs all the jobs related to reconfiguration. Whenever a membership change is initiated by the Monitoring Layer, the Group Membership layer sends the membership change message to the Atomic Broadcast layer. If a membership change message is atomically delivered by the Atomic Broadcast layer, it takes action and informs the application layer about the new view.
6.7 Monitoring

In our design choice, replica suspicion and exclusion is decoupled. Replica exclusion will not be initiated whenever there is a suspicion in the Failure Detector layer. The Failure Detector and Group Membership component don’t have the authority to make this decision. Rather a new component above the group membership layer, specifically Monitoring Layer takes such decision. This separation of responsibility provides greater flexibility [26] and high performance.

As we are using Paxos in consensus layer, the necessary failure detector $\Omega$ should elect a leader in the current view. Meanwhile the suspicion of the leader should be informed back to the Consensus layer so that the consensus doesn’t stall indefinitely in case of true failure of the leader. But a false suspicion should not lead to the exclusion of the suspected node from the replica set. So in case of a suspicion provided by the Failure Detector layer, the Monitoring layer monitors the suspected node (probably with the interface provided by the Failure Detector Layer with a large timeout value e.g. in the order of minutes) [26]. Here a suspicion will compel the monitoring layer to initiate a replica removal operation in the group membership layer. To avoid unnecessary replica removal and improve performance, the monitoring layer might consult the situation with his peer in other replicas and decide only to remove a particular replica when it learns that a threshold number of peers have suspected the same replica. Other than removing a failed replica from the replica set, Monitoring layer will also do the job of reconfiguration by handling any leaf set change events e.g. (a) Node Failure Event, (b) Node Arrival Event, (c) Leaf Set Recovered from Routing Table, (d) Leaf Set Merge, (e) Node Join Event. To make good reconfiguration it also gets information from the reliable channel layer about overflow of sending queue of a node [26, 4]. The task of reconfiguration is described in a separate section later.

6.8 Application Manager

AM is the component that we want to replicate. We have already described AM in section 1. AM is implemented as a deterministic finite automata. Whenever a message arrives from an application process or some client, the AM send it to the atomic broadcast layer. It also handles the message atomically delivered by the Atomic Broadcast layer in a deterministic way. During the implementation phase we have to make sure that any change in the AM code doesn’t compromise this property. After replication AM in vigne will look like as in Figure16.

7 Reconfiguration and Monitoring Layer

As described in section 6.7, Monitoring layer is responsible for the reconfiguration of the replica set by handling failures and new node arrivals in DHT namespace. It will issue any kind of member addition or removal operation in the current replica set. While issuing such reconfiguration commands, the monitoring layer will maintain the following invariance: (a) In case of need for a
Figure 16: Client-Application Manager-Application Process Interaction in replicated model.

Figure 17: Three Replicas in a DHT name space

remove operation (may not be a failed node) in current view the node with the most distant nodeid relative to AM-Key will be removed by keeping the best possible symmetry, (b) any new view should have majority of members from the old view to deal with unsuccessful state transfer. Because of the dynamic environment of Vigne, the reconfiguration has to deal with two major problems: (a) high churn rate, (b) routed message during reconfiguration. The purpose of this section is to introduce these problems informally and propose possible solution.

7.1 Problem with High Churn Rate

In this section, we will focus on the problem of high node arrival rate. As the case of failure has been described previously in context of failure detector, consensus and monitoring layer, we are skipping it. If there wasn’t any failure or node arrival after the job submission, three AM replicas should be running as in Figure 17. When a new node will arrive inside the replica set namespace (Figure 18), the monitoring layer might issue a membership change through the group communication layer. After the state transfer is complete, there might be four consistent replica at an instance (Figure 19). As increased number of replicas will increase the overhead of Paxos, monitoring layer will remove one.

---

†Here the size of replica set is three as the degree of fault tolerance is taken as one.

Figure 18: New Node Arrived in the DHT namespace
of the distant replica (Figure 20). But the situation might be worse. It may happen that lots of nodes are arriving in the same place of DHT and in the very next moment the scenario is like in Figure 21. So we want the monitoring layer to schedule a good reconfiguration so that the time and amount of state transfer needed to achieve the desired state in Figure 22 is minimized. More specifically we need a good heuristics for reconfiguration purpose.

One possible solution to avoid excessive and unnecessary state transfer can be as follows. The monitoring layer will keep track of node arrival and failure rates on the leaf set of the host node. If the node arrival rate is greater than a certain threshold, the monitoring layer will pause reconfiguration for a predefined or random amount of time. After the sleep period, he will issue a reconfiguration by maintaining two invariances that we have introduced. This way it will avoid some costly state transfers. But this solution has several drawbacks. It may happen that no new node will arrive after the monitoring layer has gone to sleep. Here the reconfiguration will be delayed for nothing. Besides some messages from application process or Client might have been on the fly. They will be routed to a node without any AM for stalling reconfiguration which wouldn’t happen otherwise.

7.2 Problem with routed message during reconfiguration

It is very likely that some application specific messages will be routed in the underlying structured overlay while there is an ongoing reconfiguration. So some node will receive application specific messages as in Figure 23 even though they don’t have the corresponding replica. We may hope that the monitoring layer...
will make a favorable reconfiguration such that these receiver nodes will be included inside the replica set and the queued messages will be handled. But the situation might be worse. Before the Monitoring layer has instantiated the mentioned reconfiguration, some more nodes may arrive and the current DHT namespace may look like as in Figure 24. And it may go on for couple of rounds. So the application specific messages might be scattered throughout the DHT namespace.

There are several choices to deal with this situation. We can simply discard the message by forcing application process or client to resend the message if the host node doesn’t contain the corresponding AM. This will compromise the transparency of the system and reduce the performance as well as the availability of the AM even though there isn’t any failure. Another choice is to keep the messages in a queue and send a search message to some selected node in leaf set with the AM-Key (Figure 25). If a good replica receives such message, he will send back a reply and initiate a membership change to include that node into the replica set. So the underlying reconfiguration heuristic can be ”A node receiving a application specific message will be eventually included into corresponding applications AM replica set”. But it has several drawbacks. This might lead to lots of state transfer and will reduce the performance. From this point of view we also need a good heuristic for the reconfiguration to balance
the tradeoff between reconfiguration time and message lost in highly dynamic environment.

A possible solution is depicted in Figure 25. When a node receives an application specific message, it will check whether its host node contains such replica. If the answer is yes then the processing is straightforward. If the answer is no, then he will send a search message to several neighbors in his leaf set ring with the AM-Key and IP-Address piggybacked inside the message. If some good replica receives this search message, good replica will send a reply back to the sender of search message using the IP-Address. This reply will contain the IPaddress of the current replica set. After receiving this reply, the node will re-compute its eligibility of being in the replica set. If he isn’t eligible for the replica set of the corresponding AM-Key he will forward the queued messages to good replicas using IP-Addresses. If he is still eligible he will send a join message with the queued messages to the good replicas. After receiving a join message, monitoring layer of good replicas will run a consensus and decide whether the node should be included inside the replica set or not. The result will be sent back to the node requesting join.

8 Related Work

In this section we give a short description of some projects that implemented Atomic Multicast on top of DHT.

Scribe [5] is a multicast framework developed on top of Pastry. It provides efficient application level multicast and is scalable in terms of number of groups, group size and number of messages. It uses the underlying failure detection, locality and reliability property of Pastry for group management and reconfiguration. Unfortunately it doesn’t ensure reliable and ordered delivery.
Paxon-DHT [38] is a Paxos based middleware on top of Pastry. It can support multiple services and scale well with the size of replica set and load. As it maintains virtual replicas, it fails to ensure the safety of total order delivery in all replicas. They have showed that with sufficient increases in the quorum size of Paxos consensus layer majority, the violation of safety becomes very low.

FamDHT [18] is a fault tolerant atomically accessible distributed data. It is built on top of BAMBOO DHT. It provides mutable atomic data access and provides fault tolerance by replicating data in DHT nodes. It uses Paxos to ensure that the simultaneous access to data will be executed under a total order agreement. Here the read operation can be concurrent if there isn’t any ongoing write or the system is not in synchronizing state. It uses a agreement based protocol for write commit. The system is more or less similar to our system in terms of DHT and consensus algorithm. But it is under development and they haven’t address clearly the reconfiguration issue in highly dynamic environment.

There are some other works on mutable distributed memory on top of DHT: Etna [28], RAMBO [25], and RAMBO II [16]. The work in Etna is pretty similar to our system.

9 Conclusion

Our primary goal was to increase the availability of the AM. With this goal in mind, we have investigated existing solutions in the context of Vigne. After a qualitative comparison among the various replication techniques, we have chosen Symmetric Active/Active replication. The Active/(Hot) Standby replication was eliminated due to poor latency and scalability comparing to Symmetric Active/Active replication. Then we investigated the feasibility of Symmetric Active/Active replication on top of DHT. We showed the hardship of active replication by maintaining a virtual replica set and disclosed why system like Paxon-DHT compromise consistency of replicas. Then we made the necessary specifications and design choice to implement it. We have embraced the implementation stack provided by Men-Schiper. We described each layer from the implementation point of view and made necessary design choice.

At the end we introduced two major issues during reconfiguration: high churn rate and improperly routed message during reconfiguration. We addressed the related tradeoff between costly state transfer and message lost. We proposed several solutions and discussed their relative benefits and drawbacks. Our future work will be finding good heuristic for the reconfiguration where the tradeoff between number of lost messages and reconfiguration time is balanced.

Our next work will focus on representing reconfiguration as a reinforcement learning problem. Thus the Monitoring Layer will act as an agent by maximizing its long term rewards i.e. return. In the state space of replica set, state like Figure17 will have the highest value. A state will be represented as hop distance from a reference node in the nodeid ring. The reference node will be the node whose nodeid is closest to the AM-Key on a particular moment. So the agent...
will be in state \(<1, 0, −1>\) when the replica sets are as in Figure 17. And the agent will be in state \(<1, −1, −2>\) as in Figure 18. The state of the agent will change with the dynamic environment even though the agent isn’t taking any action. The possible action space for the agent in certain state \(s\) is \(A(s) = \{\text{Replica–Removal}(\text{nodeid}), \text{State–Transfer}(\text{nodeid}), \text{Do–Nothing}\}\).

As long as there isn’t any failure or node arrival the agent will keep receiving 0 rewards in state \(<1, 0, −1>\). Our next research will focus on how to model the negative rewards in case of a failure or a node arrival or a state transfer and select a suitable algorithm to solve this problem. We believe that the agent will always come up with an optimal policy if we can model the reward according to the dynamism in DHT namespace.

In future we also want to replicate the Resource Allocator and Application Monitor, using the same specification that we have made for AM.

Acknowledgements  I acknowledge my supervisor Thomas Ropars for his constant and precious guidance during this work. I also acknowledge senior researcher Cristine Morin for her periodic and valuable feedback. I also want to acknowledge Dr. Robert C. Ward and Dr. Arif Dowla whose recommendations have helped me to get this great opportunity of working in such a nice environment.

References


