GulfStream – a System for Dynamic Topology Management in Multi-domain Server Farms

Sameh A. Fakhouri, Germán Goldszmidt,
Michael Kalantar, John A. Pershing
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY
{fakhouri,gsg,kalantar,pershng}@us.ibm.com

Indranil Gupta
Department of Computer Science
Cornell University
Ithaca, NY
gupta@cs.cornell.edu

Abstract

This paper describes GulfStream, a scalable distributed software system designed to address the problem of managing the network topology in a multi-domain server farm. In particular, it addresses the following core problems: topology discovery and verification, and failure detection. Unlike most topology discovery and failure detection systems which focus on the nodes in a cluster, GulfStream logically organizes the network adapters of the server farm into groups. Each group contains those adapters that can directly exchange messages. GulfStream dynamically establishes a hierarchy for reporting network topology and availability of network adapters. We describe a prototype implementation of GulfStream on a 55 node heterogeneous server farm interconnected using switched fast Ethernet.

1. Introduction

We describe the design and a prototype implementation of GulfStream, a distributed software system that addresses the problem of managing the network topology of multi-domain server farms. A multi-domain server farm is a collection of servers and associated hardware resources that is logically partitioned into a number of distinct network-isolated domains. Each domain in the farm establishes a unit of security, such that servers in one domain cannot access resources in another. Domains are typically established to execute a set of applications belonging to a single user or organization. For example, a single domain might be used to host a web site in a multi-domain server farm used for web hosting.

GulfStream "discovers" the physical configuration of a server farm and monitors this configuration on an ongoing basis. The discovery process is the result of a distributed algorithm, which is executed by the GulfStream daemons on all nodes. GulfStream continues to monitor the availability of network adapters, and therefore of nodes and network components, by carefully regulated heartbeating. Further, GulfStream can validate an expected topology, typically provided by a configuration database, against the discovered topology.

Traditional topology discovery and failure detection systems take one of two approaches. One approach focuses on the nodes of a cluster. In this case, failures may be repored when a it is unable to communicate over at least one of its network adapters. The second approach focuses on communication paths, concluding a node has failed only when all of its network adapters have failed. GulfStream takes the second approach. GulfStream organizes all the *network adapters* of a server farm into groups. Each group contains those adapters that are attached to the same network segment. GulfStream dynamically establishes a hierarchy for reporting network topology and availability of network adapters. Node status can be inferred by the status of the all of its network adapters.

GulfStream is also designed to directly manage the logical configuration; that is, to alter the IP addresses of network adapters, the virtual LAN configuration, and firewall settings to enforce the security and isolation requirements of domains in a server farm. Currently, GulfStream only manages virtual LAN settings, by reconfiguring the network switches via SNMP, to move servers from domain to domain. This paper does not address these control mechanisms further; they are described in [7].

GulfStream is an underlying technology of Océano [1], a prototype of a scalable, manageable infrastructure for a large server farm. This infrastructure enables multi-domain hosting on a collection of hardware resources interconnected by switched LANs. Océano is motivated by large-scale web hosting environments, which increasingly require

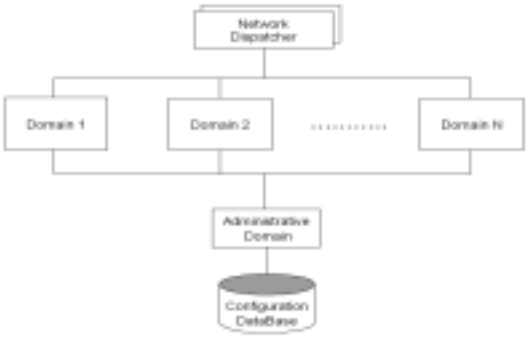

Figure 1. Topology of the Océano prototype server farm.

support for peak loads that are orders of magnitude larger than the normal steady state [14]. Océano provides a hosting environment which can rapidly adjust the resources (bandwidth, servers, and storage) assigned to each hosted web-site (domain) to a dynamically fluctuating workload.

While the hosting infrastructure uses shared servers and storage, at any point in time each resource is in use only by a single customer. Océano reallocates servers in short time (minutes) in response to changing workloads or failures. These changes require networking reconfiguration, which must be accomplished with minimal service interruption. GulfStream is the underlying technology of Océano that provides topology discovery, failure detection and topology reconfiguration functions.

Figure 1 shows the logical view of an Océano server farm. Requests flowing into the farm go through request dispatchers, for example [8], which distribute them to the appropriate servers within each of the domains. All the domains within the farm are able to communicate with the dispatchers, but are isolated from each other using private virtual LANs [5] on a switched fast Ethernet network. All domains are similarly attached to an administrative domain, where management functions reside. Management nodes are eligible to host the GulfStream view of the whole server farm.

One of the goals of Océano is to scale up to thousands of servers. The ability to scale depends on the scalability of the underlying components, including GulfStream. While no evidence of scalability is explicitly provided (there are only 55 nodes in the testbed), Section 4.2 describes the approaches used and argues that they enhance scalability. It addresses the key scaling concerns: how well groups scale both in terms of membership maintenance and heartbeating, How well the central component (the root of reporting hierarchy) for configuration verification and reconfiguration scales, and how well the use of a configuration database scales.

The next sections focus on the dynamic discovery of network topology (Section 2) and the failure detection mechanisms (Section 3). Section 4 describes the current status of the prototype implementation, presents some preliminary performance results, and discusses scalability. Finally, Section 5 provides a comparison with related work.

2. Topology Discovery

This section discusses the dynamic topology discovery protocol used by GulfStream. In a static environment, where network connections are rarely changed, the network topology could be stored in a central database, giving GulfStream all the network information it needs to start monitoring and managing the network. This approach has three drawbacks. First, there is high overhead in terms of delay and network utilization when a new node is added or a failed node is restarted. In these cases, the entire configuration must be read by the node. Second, all nodes need to have access to the topology database. The environment in which GulfStream was initially designed to operate requires domain isolation; most nodes should not have access to the configuration of all other nodes. Finally, in large environments, it is possible that the configuration database itself is incorrect unless automatically updated.

A second approach is to distribute the topology information to all the servers and have GulfStream read it locally at each node. While this approach solves the earlier problem of network overhead, insuring GulfStream daemons are working on the same version of the configuration is not practical in a large server farm. Further, problems arise when attempting to coordinate topology updates from merging network partitions and when off-line servers are returning to the farm. In particular, in Océano problems arise when a node is moved from one domain to another.

GulfStream's approach, described below, is for each node to locally discover the network topology. A small subset of the nodes, with appropriate permission, may compare the discovered topology to a central database if desired.

For discussion purposes, the configuration of a sample domain is shown in Figure 2. This example shows a layered domain with two layers, front end servers and back end servers. Other layers may be added if the domain functionality requires it. For example, a third layer may be added to create the topology of multiple server farms. The servers within the domain are shown to have multiple network adapters. The front ends are connected together with the load balancers (network dispatchers) through the adapters shown as triangles. They are also connected together and to the back end servers through the adapters shown as squares. All servers are also connected together and to the adminis-

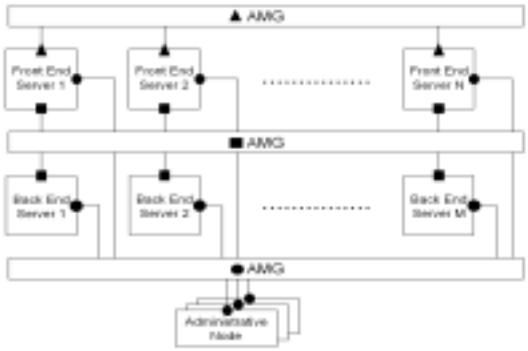

Figure 2. A single Domain in the Server Farm

trative domain through the adapters shown as circles. Note that the triangle adapters can directly communicate among themselves, but may not directly communicate with the circle adapters. The GulfStream topology discovery algorithm, described below, groups each set of connected adapters into Adapter Membership Groups (AMGs).

Note that GulfStream assumes that there are no paths between domains except for the administrative network. Further, nodes do not act as routers between the various networks to which they are connected. Access to any configuration database and the switch consoles is only through the administrative network. Lastly, GulfStream is designed for broadcast networks and makes use of hardware multicast when available.

2.1. Local Discovery

GulfStream runs on all nodes within the server farm as a user level daemon. This daemon discovers and monitors all adapters on a node. To correctly function, the daemon requires an operating system with properly initialized communication functions. Locally, the daemon obtains a list of configured adapters on its node from the operating system. The goal of the daemon is to have each of its local adapters join an AMG consisting of itself and all other adapters attached to the same network segment. To do so, each adapter begins by multicasting BEACON messages on a well-known address and port, identifying itself to all adapters that receive them. The initial beaconing phase, a configuration parameter, lasts for a few seconds, during which each daemon collects information about remote adapters (which are also beaconing). At the end of the beaconing phase, each daemon examines the collected information. If no BEACON messages were received by a particular adapter, it forms its own (singleton) AMG and declares itself the leader. In the event that other BEACON messages were received, an adapter defers AMG formation and leadership to the adapter with the highest IP address. The daemon that has the adapter with the highest IP address undertakes a two phase commit process to form the new AMG.

After the formation of an AMG, only the leader continues to multicast and listen for BEACON messages from other adapters or AMGs. This allows new adapters to join an already existing group, and for two or more AMGs that were independently formed (for example, on separate network partitions) to merge into one group. Merging AMGs are led by the AMG leader with the highest IP address. All changes to AMG membership such as joins, merges, and deaths are initiated by the AMG leader and are done using a two-phase commit protocol.

Setting the duration of the initial beaconing phase is an optimization that is dependent on the expected size of the AMGs. Setting it to zero leads to the immediate formation of a singleton AMG for each adapter. These groups then begin a merging process that would eventually end up in one AMG containing all the adapters. Forming and merging all of these AMGs is more expensive than collecting beacon messages for a few seconds. In general, group formation is rare: it occurs when the system is started [1] The cost represents a tiny fraction of the total execution time.

Once an AMG is formed and committed, the group members monitor each other using heartbeating (discussed further in Section 3). When an adapter is discovered to have failed, the AMG leader first attempts to verify the reported failure. If the reported failure proves to be correct, the AMG leader removes the failed adapter and proceeds to recommit a new group excluding the failed adapter. If the reported failure proves to be false, it is ignored. In the event that the AMG leader is the one that fails, notification is sent to the second ranked adapter (the two phase commit used for membership changes is also used to propogate membership information so that this order is known by all members) in the AMG to verify the death and become the leader of the new AMG.

2.2. Central Verification

The node that is currently acting as the AMG leader of the administrative adapters (shown as the circle adapters in Figure 2) is known as *GulfStream Central*. GulfStream Central plays a special role. It takes on three roles not held by the leaders of other AMGs. First, if available, it may access a configuration database. GulfStream Central can compare the discovered topology to that stored in the database. Inconsistencies can be flagged and the affected

[1] Currently, the full group discovery protocol may execute if group members become confused about their membership. For example, this may currently occur when multiple adapters simultaneouly fail. We expect such situtations to be eliminated as the system is refined to handle more specific failure scenarios.

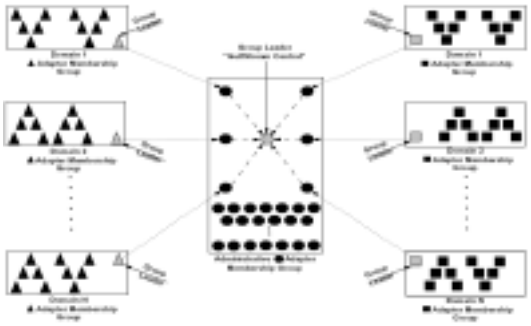

Figure 3. Hierarchy imposed on adapters by GulfStream

adapters disabled, for security reasons, until conflicts are resolved. Note that the problem caused by a central database that was identified earlier no longer applies. Here only GulfStream Central attempts to access the data instead of all GulfStream daemons. In a sense, the problem has been addressed by inverting it. Instead of reading the configuration from a database and then finding inconsistencies through discovery, GulfStream discovers the configuration and then identifies inconsistencies via the database. Second, GulfStream Central coordinates the dissemination of failure notifications to other interested administrative nodes. In this role, GulfStream Central is the authority on the status of all network components: the status of a component can be inferred from the status of all its network adapters. Because GulfStream Central has information about all adapters, it is the only node that can make such inferences. Lastly, GulfStream Central is responsible for coordinating dynamic reconfigurations of the network. This role is not explored in this paper although some of its implications are discussed in Section 3.

To support GulfStream Central's unique roles, all membership changes of the AMGs are reported, by the AMG leaders, to GulfStream Central. This is shown in Figure 3, where each AMG leader reports the group membership information through its local administrative adapter to GulfStream Central. Note that membership information is sent to GulfStream Central only when it changes. In the steady state, no network resources are used for group membership information. Further, group leaders typically need only report changes in group membership, not the entire membership. As currently implemented, GulfStream Central is centralized. It is resilient to failure in the sense that its failure results in a new leader election among the administrative adapters. This, in turn, results in a new instance of GulfStream Central. Issues of scale, as they relate to GulfStream Central are discussed in Section 4. Note also that the network assumptions, imply that network partitions will result in at most a single GulfStream Central with access to the database and the switch console(s). If nodes without such access assume the role of GulfStream central in partitions, they will be able to report failure information for that partition, but will not be able to change the network configuration.

For the leader of the administrative AMG to assume its role as GulfStream Central a node must know which of its adapters is its administrative adapter; that is through the adapter connected to the administrative VLAN. In the prototype we have developed, this is done by convention (adapter 0). However, we expect that only some nodes will be permitted to host GulfStream Central because only nodes with appropriate permissions will have access to the database and to the switches. Such nodes may contain a small configuration file granting permission to be among the nodes that may be elected leader of the administrative AMG and detailing the role of each adapter. BEACON messages sent over the administrative adapter by such nodes can be augmented with a flag indicating its role. Recipients of BEACON messages will then know that the receiving adapter must also be an administrative adapter.

3. Failure Detection

The goal of failure detection in GulfStream is to provide comprehensive monitoring of all of the communication components of the server farm, and to report when a failure is detected. Ideally, failures should be detected as soon as they occur, and the monitoring should impose negligible additional load on the servers and network segments. Unfortunately, these two ideals tend to be in conflict with each other: monitoring is generally done by sending explicit messages (heartbeats) between servers, and fast detection of failures requires that these messages be sent frequently.

This section will discuss the tradeoffs made by GulfStream to attain near-comprehensive monitoring and reasonable detection times, while minimizing the impact on the network segments. Additionally, a simple event correlation function is described, through which GulfStream infers the failure of servers, routers, and switch components.

Our basic approach to monitoring is to ensure that every network adapter on every node is capable of both sending and receiving messages. A simple approach is for the group leader of each AMG to arbitrarily arrange the adapters of the group into a logical ring, so that each adapter has a "neighbor" on its "left" and a "neighbor" on its "right". This arrangement is broadcast to the members of the AMG, and each adapter proceeds to send regular heartbeat messages to the adapter on its (logical) right and monitor heartbeats arriving from the adapter on its (logical) left. If n consecu-

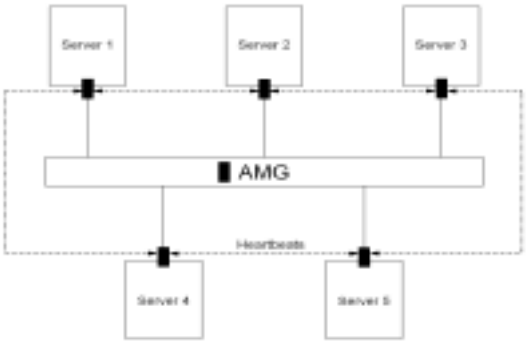

Figure 4. Heartbeating within a domain

tive heartbeats fail to arrive, a message is sent to the group leader indicating that the adapter on its left is suspected to have failed. The frequency of heartbeats (f) and the sensitivity of the failure detector (the value of n) are adjusted to trade off between network load, timeliness of detection, and the probability of a false failure report.

This scheme has two main flaws, both leading to false reports of failures. First, if a network adapter on a node fails in such a way that it ceases to receive messages from the network, this will be incorrectly "blamed" on the neighbor to the left failing to send us heartbeats. This can be ameliorated to a certain extent by first performing a loopback test on its own adapter before reporting the left neighbor as having failed. Second, this scheme is overly sensitive to heartbeats lost due to network congestion, due to its "one strike and you're out" behavior.

An improvement utilized by GulfStream is to have each adapter send heartbeats to multiple other adapters, and require a consensus of these adapters before declaring the subject adapter as having failed. This reduces the incidence of false failure reports, at the expense of additional heartbeats flowing on the network segment. A straightforward extension of the above scheme is to run the heartbeats in both "directions" around the logical ring of network adapters: each adapter sends heartbeats to both of its immediate neighbors, and monitors heartbeats arriving from each of these neighbors (see Figure 4). Both neighbors of an adapter would have to report it as "possibly failed" before it would be declared by the group leader as dead. (To handle multiple failures of "adjacent" adapters, there are instances where the group leader must investigate suspected failures on its own.)

The failures of servers, routers, and network switch components are inferred from the detected failures of the individual network adapters. This is a straightforward correlation function: if all of the adapters connected to a server are reported as failed, then we infer that the server itself has failed; likewise, if all of the adapters that are wired into a router, hub, or network switch are reported as failed, we infer that the network equipment has failed. As soon as one of these adapters recovers, we infer that the correlated node/router/switch has recovered. At present, GulfStream Central relies on a configuration database to identify how nodes are connected to routers and switches. In the future, GulfStream will independently identify these connections by querying the routers and switches directly using SNMP.

3.1. Impact of Dynamic Domain Reconfigurations

Océano creates domains by isolating groups of nodes using virtual LANs (VLANs). VLANs have the property, enforced by the switches, that adapters configured in the same VLAN can freely communicate with each other as if on the same physical network. However, adapters on one VLAN can not directly communicate with those in another VLAN (a router is needed; none is provided). Océano, to meet its quality of service guarantees, dynamically changes the membership of the domains by adding and removing nodes. It does so by reconfiguring the switches to redefine VLAN membership. In addition to changing the VLAN membership, such nodes need to change their "personality"; they may need a new operating system, applications and data. Other components of Océano [1] are responsible for these changes.

When a node is logially moved from one domain to another, the VLANs to which its adapters are connected may change. Consider one such adapter. It is not aware that the VLAN to which it belongs has changed. It still tries to heartbeat with the adapters in its original AMG. However, as a result of the move, it can no longer send and receive heartbeats with its original heartbeat partners. It concludes that its heartbeating partners have failed and attempts to inform the (original) group leader. However, it can no longer reach the group leader. Finally, it concludes that it should become the group leader and begins sending BEACON messages. The group leader of the adapter's new AMG responds to these messages, resulting in the formation of a new group. Further, the old heartbeating partners of the moved adapter are, likewise, no longer able to send messages to it and they no longer receive messages from it. They report the adapter as having failed. Their group leader recommits the group without the moved member.

Neither the leader of the old AMG nor the leader of the new AMG know that a node has moved. The old one sees the failure of a member, the new one sees a new member. The conclusion that a node has moved from one domain to another can be inferred by a third party that has access to both pieces of information: GulfStream Central. If the change is expected (in its role of reconfiguring the domains, GulfStream Central may have made the change), external

failure notifications are suppressed. If the move is not expected, it is treated as when mismatches are found between the discovered configuration and the contents of a configuration database.

4. Current Status and Future Work

A prototype of GulfStream has been implemented in Java. It is assumed that the GulfStream daemon is started on each machine when it boots, for example, via initab on UNIX machines. When it starts, the GulfStream daemon identifies all network adapters (this step is platform specific) and discovers their adapter membership groups. The adapters then participate in a heartbeating ring defined by the AMG leader. Group membership and failures are reported through the group leaders to a central component, GulfStream Central. GulfStream Central uses adapter failure notifications to infer the status of nodes. Failure notifications are published to interested parties. GulfStream Central also implements requested network reconfigurations. We have not yet implemented a complete comparison of the discovered configuration with that stored in a configuration database. This is being actively pursued. We are currently testing GulfStream on a 55 node test bed consisting of a combination of Linux (x86) and AIX (PowerPC) machines.

4.1. Performance

In principle, when networks are not heavily loaded, the time for GulfStream Central to form a stable view of the full network topology is:

$$T_d = T_{beacon} + T^{AMG}_{stable} + T^{GSC}_{stable} + \Delta \qquad (1)$$

where T_{beacon} is the duration of the beaconing phase of discovery, T^{AMG}_{stable} is the amount of time a AMG leader waits before declaring its membership stable, T^{GSC}_{stable} is the amount of time GSC waits before declaring its initial discovery to be stable, and Δ is a factor that takes into account scheduling delays on each processor. Of these parameters, T_{beacon}, T^{AMG}_{stable}, and T^{GSC}_{stable} are configurable. The graph in Figure 5 shows the time for all groups (three; each node in the testbed has three network adapters) to become stable. The graph shows the results of three experiments in which T_{beacon} was set to 5, 10, and 20 seconds. In all cases T^{AMG}_{stable}, and T^{GSC}_{stable} were 5, and 15 seconds, respectively. We expect that $\Delta = 0$. Thus, for the experiments we expect $T_d = 25, 30$ and 45 seconds. The experiments showed that for all group sizes, the time for all groups to become stable is constant as expected. However, they also show that Δ is between 5 and 6 seconds. Investigation showed that there were two main causes for this. First, nodes do not begin to beacon on all adapters before beginning the processing of other events such as received messages (these

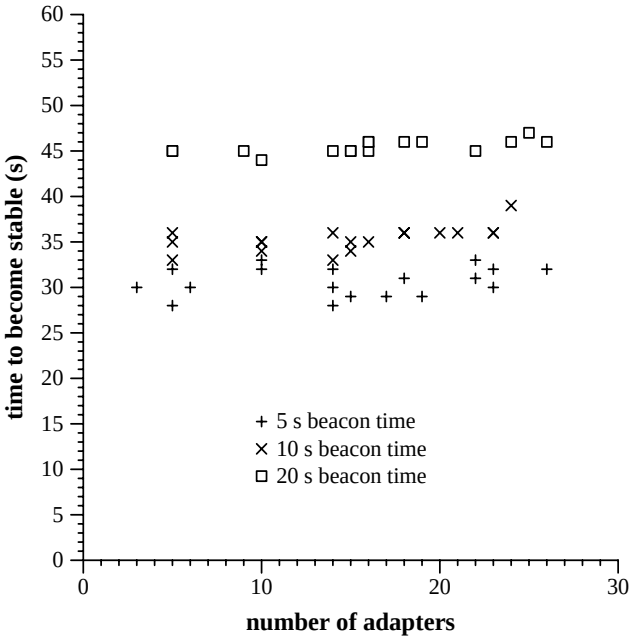

Figure 5. Time for all groups to become stable; for T_{beacon} = 5, 10 and 20 seconds.

activities are implemented as separate threads). Because of this processesing, the beaconing timer is not set for between 1 and 2 seconds after beaconing begins on the first adapter. This delay can be eliminated by ensuring that the beaconing phase is initiated on all adapters before other processing. The second element contributing to Δ is the cost of the two phase commits used to confirm group membership. This is presently implemented using point to point messages. The two phase commit can be partly optimized using multicast. These changes have not been completed and their impact not yet studied.

In some of the executions studied, not all of Δ was accounted for by these two elements. We suspect that the remaining time was spent in thread switching. GulfStream is in implemented in Java using multiple threads. Threads are temporarily created to handle specific logical tasks. No special effort was made to give GulfStream priority in execution. Hence, Δ includes time spent swapped out (other processes are executing) and in switching between threads. It is necessary to verify this and consider its implications when nodes are heavily loaded.

When the networks are heavily loaded, there is a possibility that a node will miss all of the BEACON messages issued during a beacon phase. Assuming independent loses, if p_n is the probability of losing a message in the network, then the probability of losing m BEACON messages is $(p_n)^m$. In this case, an initial topology will still be formed in T_d time; however, some nodes will be missing. We have

not yet further studied the distribution of missing nodes in the initial topology as a function of network load.

4.2. Scalability

While the analysis and data presented above indicates that discovery requires a constant time for different numbers of nodes, it fails to support a claim of scalability beyond 55 nodes. Scalability is important because it translates into (1) a larger customer base for the server farm, (2) a potentially large number of servers for any particular customer to use whenever load demands it, and (3) a lower percentage of administrative machines in the farm as compared to servers doing useful work. In the next few paragraphs, we describe both the design approaches taken in GulfStream to enhance the likelihood of its being scalable and we discuss several alternative designs that could be used if necessary.

There are three scaling concerns with GulfStream. First, how well AMGs scale both in terms of membership maintenance and heartbeating. Second, how well the central component for configuration verification and reconfiguration scales. Lastly, how well the configuration database scales. We address each in turn.

Adapter Membership Groups. Past experience [4, 10] suggests that the key element limiting scalability of topology discovery algorithms is the sizes of the messages exchanged and the key limiting factor for failure detection scalability is the frequency of heartbeating messages.

The topology of a multi-domain server farm is such that a whole farm is divided into a number of (network) isolated domains. GulfStream takes advantage of this hierarchy by allowing each adapter to locally discover its neighbors. Only one is selected to interact with a higher level. In the current prototype, there are only two levels. However, this hierarchy could be extended. Further, both the number of messages and content of messages is limited. Only AMG leaders send messages about membership. They forward only membership changes to GulfStream Central. Finally, the discovery phase is not open ended; there are fixed time limits on it. Later, disjoint groups can be merged into a larger group.

Currently, GulfStream identifies adapter failures by having each adapter heartbeat its (logical) neighbors. While such a scheme works well for a small number of adapters in the ring, the rate of failures among even a few hundred adapters in the ring may impose undue instability and overhead on the group leader in maintaining the ring itself, resulting in a degradation in performance. One interesting alternative is to divide each (large) AMG into several small subgroups, with all members within one subgroup tightly heartbeating only each other. This effectively decreases the size of AMGs. In this case, the group leader (of the whole group) needs to poll the status of each subgroup, at a very low frequency, to detect the rare event of a catastrophic failure of all members in a subgroup. Such a scheme would ensure that the performance of GulfStream is not degraded in the event of more than one failure at a time in the system. Moreover, the subgroups could be formed based on the physical network topology, reducing the heartbeat load on the network. A radically different approach to failure detection is to eliminate heartbeating altogether and use a randomized distributed pinging algorithm among group members. While such a protocol sounds too simple and trivial to result in better performance, initial calculations show otherwise: protocols in this category impose a much lower load on the network compared to heartbeating protocols that guarantee the similar detection time for failures and probability of mistaken detection of a failure [9].

GulfStream Central. To ensure scalability of GulfStream Central, its function can be distributed. While this would ameliorate the problem of heavy infrastructure management traffic directed to and from a single node, and the resulting degradation in performance, it would make the programming of GulfStream Central that much more difficult. Moreover, it is currently not clear how much data sharing a distributed version of GulfStream Central would need. A lot of data sharing makes distribution less attractive because of the cost of the data sharing. At present a wait and see attitude is being pursued; a decentralized approach will be used if the experimental overhead suggests that it is necessary to achieve desired scaling.

Configuration Database. Scaling concerns for the configuration database can be addressed by two, not necessarily disjoint, possibilities: 1) use a distributed database, and 2) use a storage area network (SAN). Distributed databases have the advantage of high availability, while using a SAN to access the same provides high-speed connection to the database. Besides the configuration data, a SAN component could be used to store other data as well. As mentioned in Section 2, access to the configuration database has been limited to GulfStream Central. To a great extent this permits a larger farm before the database becomes a scaling bottleneck.

5. Related Work

Most of the work on managing large-scale distributed networked systems couple message delivery semantics with failure detection and recovery semantics. The most prominent abstraction in this realm is the concept of *Virtual Synchrony*, as pioneered by the Isis project [2]. Here, all the processes that belong to a particular group are guaranteed to see all group multicast messages and membership changes

(due to the joining and leaving/failure of members) in the same order, regardless of the unreliability of the underlying network. This has been proved to be unachievable in an asynchronous fault-prone network [3]. Most implementations of Virtual Synchrony (eg., [2, 6, 10]) get around this impossibility by partitioning out the group into smaller groups until Virtual Synchrony can be guaranteed within each of these. Thus, in the worst case, after a string of failures, each of the original group members ends up in a group of its own.

However, we wish to emphasize here that GulfStream is aimed at solving a problem that is weaker than Virtual Synchrony. More specifically, GulfStream does not need to guarantee Virtual synchrony among the adapter groups: these groups exist solely for the purpose of detecting failures and reporting them to central authority (GulfStream Central).

Our ring protocol to detect failures is similar to that of the unidirectional ring protocols used in the Totem system [13] to guarantee consensus and reliable ordering of messages at the LAN and WAN level, as well as the bidirectional ring protocols for failure detection used in the High Availability services in the IBM RS/6000 SP [12, 4]. In fact, the guarantees provided by the GulfStream system are comparable to the "strong" guarantees described in [12], without the overhead of reporting membership changes to all participating processors (since only GulfStream Central needs access to this information).

The primary difference between GulfStream and the SP's HATS service [4] is that the former manages the topology of a cluster built from off-the-shelf components, while the latter is restricted to a homogeneous environment connected with a special purpose high speed interconnection network.

HACMP[11] uses a form of heartbeating which scales poorly. Further, HACMP, like GulfStream, integrates features for network reconfiguration into the infrastructure. Unlike GulfStream, this reconfiguration is largely manual.

Another service that seeks to manage clusters of workstations is the Microsoft Cluster Service [15]. The Microsoft Cluster Project, however, is set in a workstation-type environment (as against the server farm-type of environment we are dealing with), and does not scale beyond a few tens of workstations [15].

6. Acknowledgements

Jun Fong, Tim Jones and Terry Steilan provided us with an understanding of VLANs and their implementation on the Cisco 6509 switch. The whole Océano team provided feedback and the context in which to build GulfStream. Jehan Sanmugaraja, Jim Norris and Michael Frissora spent many hours supporting the hardware and network in our test farm.

References

[1] K. Appleby, S. Fakhouri, L. Fong, M. K. G. Goldszmidt, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger. Océano — sla based management of a computing utility. In *Proc. of the 7^{th} IFIP/IEEE Intl. Symp. on Integrated Network Management (IM 2001)*, pages 855–868, may 2001.

[2] K. P. Birman. The process group approach to reliable distributed computing. *Commun. ACM*, 36(12):37–53,103, December 1993.

[3] T. Chandra, V. Hadzilacos, S. Toueg, and B. Charron-Bost. On the impossibility of group membership. In *Proc. of the 15^{th} Annual ACM Symp. on Principles of Distributed Computing*, pages 322–330, May 1996.

[4] Chiakpo and et al. *RS/6000 SP High availability infrastructure*. IBM, November 1996. Red Book.

[5] CISCO. *CISCO 6509 Switch Documentation*.

[6] D. Dolev and D. Malki. The transis approach to high availability cluster communication. *Commun. ACM*, 39(4):64–70, April 1996.

[7] J. Fong and M. Kalantar. Isolated dynamic clusters for web hosting. In preparation, 2001.

[8] G. Goldszmidt and G. Hunt. Scaling internet services by dynamic allocation of connections. In *Proc. of the 6^{th} IFIP/IEEE Intl. Symp. on Integrated Network Management (IM 1999)*, pages 171–184, 1999.

[9] I. Gupta, T. Chandra, and G. Goldszmidt. On scalable and efficient failure detectors. In *Proc. of the 21^{st} Annual ACM Symp. on Principles of Distributed Computing*, 2001.

[10] IBM. *Group Services Programming Guide and Reference*. Order Number SA22-7355.

[11] IBM. *HACMP for AIX 4.4 Concepts and Facilities*. Order Number SC23-4276-02.

[12] F. Jahanian, S. Fakhouri, and R. Rajkumar. Processor group membership protocols: specification, design and implementation. In *Proc. 12th Symp. on Reliable Distributed Systems*, pages 2–11, October 1993.

[13] L. E. Moser, D. Agarwal, R. K. Budhia, and C. Lingley-Papadopoulis. Totem: a fault-tolerant multicast group communication system. *Commun. ACM*, 39(4):54–63, April 1996.

[14] M. Squillante, D. Yao, and L. Zhang. Web traffic modeling and web server performance analysis. In *Proc. of the 39^{th} IEEE Conf. on Decision and Control*, December 1999.

[15] W. Vogels, D. Dimitriu, A. Agarwal, T. Chia, and K. Guo. The design and architecture of the microsoft cluster service — a practical approach to high-availability and scalability. In *Proc. of Symp. on Fault-tolerant Computing Systems (FTCS'98)*, June 1998.