The Galaxy Framework for the Scalable Management of Enterprise-Critical Cluster Computing[†]

Werner Vogels, Robbert van Renesse, Ken Birman

Dept. of Computer Science, Cornell University

Abstract

In this paper we present the main concepts behind the Galaxy cluster management framework. Galaxy is focused on servicing large-scale enterprise clusters through the use of novel, highly scalable communication and management techniques. Galaxy is a flexible framework built upon the notion of low-level management of cluster farms, where within these farms islands of specific cluster management types can be created. A number of cluster profiles, which describe the components used in the different cluster types, are presented, as well as the components used in the management of these clusters.

1 Introduction

The face of enterprise cluster computing is changing dramatically. Where as in the past clusters were dedicated resources for supporting particular styles of computing (OLTP, large batch processing, parallel computing, high-availability)[8,13,14,15,24,25], modern Data Centers hold large collections of clusters where resources can be shared among the different clusters or at least easily re-assigned to hotspots within the overall Data Center organization. A Data Center may see a wide variety of cluster types; cloned services for high-performance document retrieval, dynamic partitioning for application controlled load balancing, hot-standby support for business-critical legacy applications, parallel computing for autonomous data-mining and real-time services for collaboration support.

It must be obvious that such a complex organization with a variety of cluster types exceeds the scope of conventional cluster management systems. In the Galaxy project we are concerned with constructing a framework for data-center-wide cluster management. Using this framework a management organization can be constructed that controls clusters and cluster resources in an integrated manner, allowing for a unified, data-center-wide approach to cluster management.

Galaxy provides a multi-level approach. In its most basic configuration this offers two abstractions; the first is the *farm*, which is the collection of all nodes that are managed as a single organization, and the second is a *cluster* which is an island of nodes within the farm that provides a certain cluster style management, based on a *cluster profile*. A profile is a description of a set of components, implementing advanced distributed services that make up the support and control services for that particular cluster.

[†] This research is supported by the National Science Foundation under Grant No. EIA 97-03470, by DARPA/ONR under contract N0014-96-1-10014 and by grants from Microsoft Research

Extreme care has been given to issues related to scalability, especially with respect to the components implementing the communication and distributed control algorithms. The novel techniques used in the communication, membership and failure detection modules allow scalability up to thousands of nodes. In our limited experimental setup we have shown that scaling up to 300-400 nodes is possible, giving us confidence that our future experiments with even larger sets of nodes will continue to show excellent scalability.

The first phase of the development of Galaxy is complete; the framework and a set of generic management components are implemented and a number of example cluster profiles and their specialized management components are in daily use. The system is developed for the Microsoft's Windows 2000 operating system, integrating tightly with the naming, directory, security, and other distributed services offered by the operating system. Galaxy has been selected by a major operating system vendor as the basis for its next generation cluster technology.

This paper is organized as follows: in sections 2 and 3 the model underlying the framework is presented and in sections 4 and 5 we provide some background on our approach to building scalable distributed components. In sections 6 through 9 details are given on the design of the farm and cluster support. The paper closes with some references to related work, a description of our upcoming distribution, and plans for the immediate future.

2 General Model

Galaxy uses a multi-level model to deliver the cluster management functionality. The basic abstraction is that of a *farm* [8], which is a collection of managed machines in a single geographical location. The farm can consist of potentially thousands of nodes, and is heterogeneous in nature, both with respect to machine architectures and network facilities. Galaxy provides a set of core components at each node in the farm that implement the basic control functionally.

The administrative personnel responsible for the overall operation of the farm has a set of farm management tools, within which the *Cluster Designer* is most prominent. This tool allows an administrator to construct islands of *clusters* within the farm, grouping nodes in the farm together in tighter organizations to perform assigned functionality. Different cluster styles are defined using *Cluster Profiles*, which describe the set of components that make up the management and application support functionality for that particular cluster style. A new cluster is created based on a profile, and when a node is added to the cluster it will automatically instantiate the components described in the profile for the role this particular node is to play, and join the cluster management group for purpose of intra-cluster failure monitoring and membership management.

Both at the farm and cluster level, the system architecture at a node is identical. At the core one finds a management component that implements membership, failure detection and communication. This management service is surrounded with a set of service components implementing additional functionality, such as an event service, a distributed process manager, a load management service, a synchronization service and others. These components export a public service specific API, to be used by other management services or applications. Internally each of these components implements a

standard service API so that the components can be managed by the farm or cluster management service. The components receive service specific membership information, indicating the activity status of this service at the other nodes in the cluster or the farm.

At a node that is part of a cluster, one would see two management services: one for the farm and one for the cluster, each with its own set of managed service components. To correctly visualize the hierarchy one should imagine an instance of the Cluster management service as a component managed by the Farm management server. In principle a node can be part of multiple clusters, although we have only found one case in which overlapping was practical. We believe however that if creation of clusters as lightweight management entities is a simple enough action, we may see scenarios in which overlapping does play an important role.

The existing case where overlapping clusters is used in Galaxy arises when grouping a collection of nodes into a management cluster. This cluster runs a set of services that is specific for controlling the overall management of the farm and configured clusters. The nodes in this cluster can be dedicated management nodes or can be part of other clusters, performing some general management tasks as background activities. The services that are specific for the management cluster are mainly related to event collection, persistency of overall management information and the automatic processing of management events.

2.1 Extensions to the basic model

After Galaxy had been in use in production settings the need arose for a two additional management abstractions that were not targeted in the initial design of Galaxy. The first extension, the notion of a *meta-cluster*, was introduced to deal with the limited scalability of legacy cluster applications. The second extension, *geoplex management*, addresses the need to have cluster management structures span nodes in multiple farms, where the farms are geographically distributed.

Even though the distributed systems technology underlying Galaxy is highly scalable, the technology on which legacy applications are based often is limited in scale. Ideally the application cluster would consisted of all the nodes of necessary to implement the functionality, but the limited scalability forces the cluster designer to break the cluster into a set of smaller cluster based on the scale of the cluster application. To still be able to consistently manage the resources shared among these smaller clusters, Galaxy implements a notion of a *meta-cluster* which provides meta-cluster wide membership and communication primitives.

To manage clusters in a *geoplex* setting, Galaxy was augmented with a technology similar to the meta-clusters, but which functions at the farm level instead of the cluster level. In a *geoplex* membership entities are farms, and failure-detection and membership tracking is performed at the level of complete farms. Through geoplex-wide communication primitives the farms share node and cluster level membership information.

Galaxy does not allow clusters to span nodes in multiple farms; however it does allow meta-clusters to encapsulate clusters that are in different farms to provide management of functionality offered through geographical distribution.

3 Distribution Model

Distribution support for clusters comes in two forms: first there are the distribution services to support the farm and cluster management infrastructure itself, these services provide the core mechanisms for all the distributed operations in Galaxy. Secondly there is the support offered to the applications that run on the clusters. In Galaxy the latter is offered through the services that are unique for each cluster configuration.

A very natural approach in structuring these services is to model them as groups of collaborating components [2,3]. This notion of groups naturally appears in all many places in the system, whether it is in naming, where one wants to be able to address the complete set of components implementing a service, in communication where one wants to send messages to all instances of a service, or in execution control where you want to synchronize all service instances, etc. The notion of a group appears from the core communication level all the way to the highest abstract level where one wants a farms and cluster to be viewed as groups to be able to access and address then as single entities.

In Galaxy the *process group* model is used throughout the whole framework, both in terms of conceptual modeling of the system, as in the technology used to provide the distribution support. In a modern architecture such as Galaxy, groups no longer necessarily consist of processes but are better viewed as interacting components.

The technology that allows us to design the distributed components using a group abstraction provides at its core a view of the components in the group through a *membership service*, identifying which instances are presently collaborating. The service uses a *failure detector* to track the members of the group and to notify members of changes in the membership. The membership also provides a naming mechanism for communication, both for communication with the complete group as well as with individual members [4]. Modeling collaboration components as a group allows us to build support for complex interaction patterns where, for example message atomicity, request ordering or consensus on joint actions, play an important role. This core functionality is used to implement the various distributed operations such as state sharing, synchronization, quorum based operations, etc.

The communication and membership technology used to provide the group abstraction as part of Galaxy is a crucial component in achieving scalability. In Galaxy a fourth generation group communication system is used, where most of the limits on scalability that these systems exhibited in the past have been overcome.

4 Scalability

Cornell's Reliable Distributed Computing group has been building communication support for advanced distributed systems for the past 15 years, resulting in systems such as the Isis Toolkit and the Horus and Ensemble communication [4,20] frameworks. Most recently, the Spinglass project focuses on issues of scalability in distributed systems, with early results in the form of highly scalable communication protocols [6], failure detection [19] and resource management [21]. The scalability vision that drives the research in Spinglass, also has driven the design and development of Galaxy. The research has transitioned into companies such as Isis Distributed Systems and more recently into Reliable Network

Solutions, Inc., which provides support for building industrial strength distributed systems based on the Spinglass research results.

Building on our experiences of building distributed systems, mainly in the industrial sector, we have collected a set of lessons that are crucial to developing scalable systems. These lessons drive our current research and are applied in the design of Galaxy project.

The five most important are:

1. *Turn scale to your advantage*. When developing algorithms and protocols you have to exploit techniques that work better when the system grows in scale. Adding nodes to a system must result in more stable overall system and that provides more robust performance instead of less. Any set of algorithms that cannot exploit scaling properties when a system is scaled up is likely be the main bottleneck under realistic load and scaling conditions.

   Successful examples of this approach can be found in the gossip-based failure detector work [19], the bimodal multicast protocols [6], and the new multicast buffering techniques [16].

2. *Make progress under all circumstances*. Whenever a system grows it is likely that, at times, there will be components that are experiencing performance degradations, transiently or permanently, or even may have permanently failed or brought off-line. Traditionally designed cluster management systems all experience an overall performance degradation whenever certain components fail to meet basic performance criteria, often resulting in a system that runs at the pace of its slowest participant. These dependencies need to be avoided at the level of the cluster management infrastructure to avoid scenarios in which bad nodes in the system drag down the complete cluster or, at the worst case, the farm [5].

   Systems built based on the epidemic technology provide a probabilistic window within which the system is willing to tolerate failing components, without affecting the overall system. If, after this time window, the component has not recovered it is moved from the active set to the recovery set, where system dependent mechanisms are deployed to allow the component to catch up. This minimizes the impact on the overall system performance [4].

3. *Avoid server-side transparency*. If applications and support systems are designed in a manner unaware of the distributed nature of the execution environment, they are likely to exhibit limited scalability. Achieving true transparency for complex, production quality distributed systems is close to impossible, as the past has shown. Only by designing systems explicitly for distributed operation, will they be able to handle the special conditions they will encounter and thus be able to exhibit true scalability [29].

4. *Don't try to solve all the problems in the middleware*. As a consequence of a misguided attempt to guarantee transparency, research systems have focused providing support for all possible error conditions in the support software, attempting to solve these without involving the applications they need to support. The failure of this approach demonstrates a need for a tight interaction between application and support system. For example applications can respond intelligently to complex conditions such as network partitioning and partition repair [30].

5. *Exploit intelligent, non-portable runtimes.* Cross platform portability is a laudable goal, but to build high performance distributed management systems one needs to resort to construct modules that encapsulate environment specific knowledge. For example the application must be able to inspect the environment using all available technology, and use the resulting knowledge in an intelligent manner. Any system that limits itself to technologies guaranteed to be available cross-platform, condemns itself to highly inefficient management systems, that are unable to exploit platform specific information which is necessary in achieving high-performance.

   An important example is failure detection of services, hosts and networks. Approaching this problem in a cross-platform, portable manner is likely to result in a system that can only exploit a heartbeat mechanism, possibly augmented with some SNMP status information [26]. Using specific knowledge about the environment will allow for construction of highly efficient failure detectors. A simple example is that some systems make ICMP error information available to sender of the UDP message that triggered the error response, often an indication that the recipient is no longer available, and the failure detection can be cut short. Another example arises in the case where nodes are connected through a VIA interconnect, which provides ultra-reliable status information about the interconnected nodes, providing failure detection without the need for nodes to communicate.

5 Distribution Support System

The Galaxy framework is designed with the assumption that there is access to a high-performance communication package, which provides advanced, configurable distributed systems services in style of the Horus and Ensemble systems. These services have to exhibit the scalability properties needed to provide a solid base for a scalable management system, as described in the previous section. In the current implementation we use an advanced communication package based on the new Spinglass protocols.

The *Scalable Group Communication Service* (SGCS) package is similar to the Horus and Ensemble in that it is built around a set of configurable communication stacks, which implement a unified mechanism for point-to-point and group communication, and provides an abstraction for participant membership and failure detection. This new system represents an advancement over these previous research systems in that it is based on the scalable failure detection, communication and state sharing protocols developed in the Spinglass project, and uses a next generation stack construction mechanism, allowing for asymmetric, optimized send and receive paths, and a message management paradigm that provides integration with user-level network facilities.

The communication system provides mechanisms for the creation of communication of multi-party communication channels with a number of reliability and ordering guarantees, and offers a unified name space for group and point-to-point communication.

The protocol core of the system is portable but its runtime is specifically adapted to function well in kernel environments, and parts of the system used in this research runs as a kernel module in Windows 2000. The complete package is encapsulated as a component server integrating well with existing development tools.

5.1 Failure Detection and Membership Services

Essential for a reliable distributed system is the knowledge of which participants are available and which are off-line or have failed. To achieve this most systems deploy a failure detection service that tracks nodes and reports possible failures to subscribers. In Galaxy a multi-level failure detection service is used that can be configured to respond to failures in different time-frames. This gives us the opportunity to make trade-offs between accuracy, speed of detection and resource usage. For example in a cluster that runs a cloned application (i.e. a web server with replicated content), the reconfiguration triggered by a false suspicion is simple and can easily be undone when the system recognizes the mistake. However in the case of a partitioned database server a failure notification triggers a reconfiguration of the database layout and is likely to pull a standby server into the cluster configuration. Such a false suspicion is a considered a disaster.

The multi-level failure detector has several modules, of which the most important are:

5.1.1 Gossip-based failure detector

The failure detector is part of the *light-weight state service* of SGCS, which provides functionality for participants to share individual state, the consistency of which is guaranteed through an epidemic protocol. A node includes in its state a version number which it increments each time is contacts another node to gossip to. In this gossip message the node will distribute a vector of node identifiers combined with the latest version numbers and a hash of the state information it has. The receiver of this message can decide, by comparing the version number and hash vectors, whether it has information that is newer than the sender, or that the sender has information that is newer. The receiver can then decide to push the newer information is has to the original gossiper, and to request transmission of the updated information from the gossiper. Failure detection can be performed without the need for an additional protocol: whenever a node receives a version number for a node that is larger than the one it has, it updates a local timestamp for that node. If for a certain amount of time the version number of a node has not been incremented, the node is declared to have failed. The rigorous mathematics unpinning the epidemic data dissemination theory allows for us to exactly determine the probability that a false suspicion will be made based on the parameters such as gossip rate and group size. As such this approach gives a clear control over the accuracy of the failure detection. A second advantage of this technique is that it allows each node to completely autonomously decide on whether another node has failed without the need communication with other nodes.

5.1.2 Hierarchical gossip-based failure detector

Even though the gossip techniques are highly scalable, to minimize the communication and processing load needed to run the failure detector, a number of additional techniques are used. The SGCS light-weight state service has a notion of distance between nodes, which is partly based on operator configuration and partially determined through network

tracking. The service will gossip less frequently with a node when the distance to that node increases. This has the additional advantage that routers overload will not have an averse on the overall distribution probability. A second approach which yields an extremely scalable service is to organize the nodes into zones and organize the zones into virtual information hierarchies. The non-leaf zones in the hierarchy do not contain node information, but information on the child zones which are summaries from the individual entries at that zone level. This technique is successfully deployed in the Astrolabe tools and will used for further integration into Galaxy [22].

5.1.3 Fast failure detector.

The gossip based failure detector is highly accurate, but slow. Over common communication links, with the communication load lower than 1% and a group size up to a thousand nodes, Galaxy tunes the failure detection threshold to be around 7-8 seconds. This is convenient as a number of anomalies in PC and operating systems architectures have shown that there are scenarios under which nodes can be network-silenced for several seconds, which would cause false suspicions when using more aggressive thresholds. Experiments with gossip-based failure detectors with more aggressive thresholds have been successfully performed, but mainly in controlled hardware and communication settings [18].

To address the need for faster failure detection, modules have been added to the multi-level failure detector that use more traditional techniques to track other nodes. These techniques are not as general and scalable as the gossip-based detector, and as such are only deployed on subsets of nodes. Most commonly used is the buddy failure-detector module where in a group the individual nodes will *ping* each other and the failure of a node to respond to pings will trigger a failure suspicion. This suspicion decision is broadcast to the other nodes. This module is fully configurable in how many nodes a single node will track and how many suspicions one needs to receive before the node will indicate the suspicion to the other system modules. The module makes use of the membership information produced by the light-weight state service to configure which nodes to track.

In Galaxy the farm membership is based on the gossip-based failure detection techniques, while the fast failure detectors are added to nodes based on management configuration. This can be based on knowledge of specific node and network configurations, or it can be part of a specific cluster profile.

5.1.4 Environmental failure detectors

The multi-level failure detector has a plug-in architecture for a class of very specialized failure detectors. These modules can track the environment to make intelligent and often highly accurate decisions about node availability. Examples of modules developed for particular Galaxy deployments are

- UPS management-event processors, which using the power supply information to determine the health of a node.
- SNMP monitors of switch link state information
- ICMP error message processors
- High-speed interconnect connection-state information

5.1.5 Consensus based membership

To support environments where agreement on the membership is essential, SGCS provides a consensus based membership protocol that runs a traditional leader-based consensus protocol. This membership module takes the suspicions generated by the fast or gossip-based failure detectors and runs an agreement protocol to ensure that all the participants in the group have seen the membership changes and have taken actions accordingly. If the SGCS stack instance is configured which virtual synchronous messaging, it will tag information onto the membership agreement messages to ensure agreement on message delivery with respect to the membership changes.

5.1.6 The use of different membership views in Galaxy

There are five different membership views in Galaxy, each of them implemented using some or all of the failure detectors described above.

1. *Farm Service membership*. This membership is based on the gossip based failure detector, and the cluster administrator configures the threshold of when to switch to the hierarchical gossip based failure detector. The cluster administrator also has the option to add fast and environmental failure detectors to the farm service but in general the gossip-based failure detector is sufficient to implement the farm service membership functionality.
2. *Cluster Service membership*. This service uses the membership information from the farm service as basic input and adds at least a fast failure detector to run within the cluster group, possibly augmented with the consensus based membership module. The exact modules to be used, including the additional environmental modules, are based on the cluster profile constructed by the administrator.
3. *Component membership*. The cluster service maintains the membership list for the local galaxy components that subscribe to the cluster service functionality, and exchanges this information with the other nodes in the cluster. Given that this shared state information is of the single-writer, multiple-reader kind, no special communication properties are required except for a reliable group multicast. When a new node joins the cluster it receives a state message from each of the other cluster nodes. The components receive a membership view which includes the state of each cluster node, information on which other nodes are running this particular component.
4. *Meta-cluster membership*. Each of the clusters in the meta-cluster group elect two nodes that will join a separate communication channel on which they will the exchange membership information from their local cluster. Associated with this communication channel is a failure detection module that runs less aggressive than the failure detectors in the member clusters. If at the local cluster the failure of an elected node is detected, a new node is elected to join the meta-cluster membership, before the meta-cluster failure detector is triggered. If the meta-cluster failure detector is triggered it is an indication that all the nodes in a member cluster have failed.
5. *Geoplex membership*. This membership protocol uses a technique similar to the meta-cluster. Each farm elects a number of nodes to participate in the geoplex membership protocol, where upon node failure new nodes are elected to

maintain the membership. The module is in general augmented with an environmental failure-detector that tracks the availability of the communication links between the data-centers.

5.2 Epidemic communication

Although a detailed description of the protocols used in SGCS is outside of the scope of this paper, it is important to understand the nature of the new protocols used to support the Galaxy operations. SGCS relies heavily on so-called epidemic or gossip protocols. In such protocols, each member, at regular intervals, chooses another member at random and exchanges information. Such information may include the “sequence number of the last message received", the "current view of the group", or the "version of the light-weight state”. These gossips are known to spread exponentially fast to all members in spite of message loss and failed members, and therefore scales extremely well. In fact, this dissemination process is a well-understood stochastic process about which we can make several probabilistic guarantees. It allows the protocol designer to provide guarantees such as “the probability that a message is not delivered atomically is less than epsilon”. Here, epsilon can be configured to be a very small, configurable constant.

Next to the excellent scalability of the information dissemination techniques, protocols based on these techniques are also very robust to node failures and message loss. Once information has been exchanged with a few nodes, the probability that it will not reach all nodes eventually is very small. Experiments with 25% of message loss, or with node perturbed 50% of the time, show that it is almost impossible to stop the information dissemination once a few nodes have been infected.

In SGCS the light-weight state service and the basic reliable multicast are implemented using epidemic protocols. These group membership protocols provide views to group members, which in turn, are built using information from the light-weight state service, and are thus also probabilistic in nature. On top of these protocols, SGSC provides lightweight versions of protocols that give more traditional (non-probabilistic) guarantees, such as virtual synchrony, total order and consensus.. Because these protocols can now rely on the high probability of success, the amount of work they have to put into buffering, for example, is drastically reduced. These protocols do need to be aware that there are conditions under which the guarantees for success will be violated (i.e. a message will not be delivered or recovered by the reliable multicast protocol). These conditions however are detected by the lower-level protocols and indicated to its subscribers, which can then run recovery protocols if necessary.

6 The Farm

Essential to enterprise cluster management is that the set of nodes out of which the clusters are created, can be viewed and managed as a single collection. This provides the manager with a single access point through which the operation of the complete data center clustering can be viewed and controlled. At this level Galaxy provides the notion of a *Farm*, which brings together all managed nodes into a single organizational unit.

To provide the farm abstraction, each node runs a *farm service*, which implements farm membership, node failure detection, intra-farm communication and state sharing. Additionally the farm service includes configuration and security services, as well as management functionality for control of a collection of *farm service components.* These components are separate from the farm service process, allowing for the set of services used in a farm to be configurable, and extendable. Farm service component must implement a service control interface used by the farm service process to control the service components and to establish a communication mechanism for components to communicate or exchange state.

The service components that are generally loaded at each farm node are:

Event Service: this service monitors the state of local node and generates *Farm Events,* which are disseminated to the collection of nodes that make up the management cluster. At the management cluster the events are automatically processed, and saved for off-line auditing. The event processors in the management cluster have the ability to find correlations in generated events, and to generate operator warnings. Future work in this area will include a capability for the event processors to automate corrective actions for certain alarm conditions. We are looking for integration of our event technology with current advances in online data mining. The Event service is extensible and can use a variety of sources from which events can be drawn; current mechanisms include the Windows2000 Event log, WMI objects, performance counters and ODBC data sources. The set of information sources to monitor, as well which objects to collect from the sources is configurable. The service distributes its events in an XML encapsulation, allowing for easy extending at both event generator end process sides.

Measurement Service: This service collects local performance information and posts this information in the Farm Management Information Database. The type and amount of information, as well as the update frequency is configurable at runtime when the information is drawn from the WMI and performance counters collections.

Process and Job Control Service: This service implements functionality for controlled execution of processes on farm nodes. The service provides full security and job control, through an interface that allows authoritative users to create jobs on sets of nodes, add processes to an existing job and control the execution of the job. It provides full flexibility in client access. A client does not necessarily need to connect to a node where it wants to start processes, but can connect to any node in the farm to create and control jobs on any other node.

Remote Script Service: Similar to the process service is the remote scripting engine, which allows administrators to produce simple scripts and execute these scripts concurrently on sets of nodes. This service is based on the Active Scripting component, which is augmented with a set of simple synchronization services, and output redirection and logging services.

Component Installation Service: This is a simple service that ensures that the current components versions are installed based on farm configuration information and assigned cluster profiles.

6.1 Inserting nodes in the Farm

To add a node to the farm, an administrator installs the basic farm management service and components, and starts the farm service. The farm service at the node uses a network resource discovery mechanism at startup time to locate basic information about the farm it is to be active in; it then announces itself to the farm controller service, which is part of the management cluster. The node will not become part of farm automatically, but instead its insertion request is queued until an administrator has confirmed that this node is granted access to join the farm. Included in the confirmation process is a set of security actions that will enable the new node to access security tickets necessary for running a farm service, and to retrieve the necessary key information to communicate with the other farm members. After the new node has joined the farm it retrieves the farm configuration, which results in the launch of the service components.

6.2 The Farm Information Database

Information about the nodes in the farm is kept in the Farm Management Information Database. In part this database is filled with static, administrator supplied information about the nodes, but much of the information in the database is dynamic in nature and updated constantly by the farm management components such as the measurement and the process control services. The database is completely distributed in design and can be accessed from each of the nodes in farm without the need to contact other nodes. The database is updated using an epidemic dissemination protocol to ensure scalable operation, automatically controlling the update rates and the amount of data transferred based on the number of nodes involved.

6.3 Landscapes

To manage very large farms Galaxy provides a mechanism for viewing the state of the farm using different *Landscapes*. Landscapes are employed as a mechanism for organizing nodes into smaller collections, such that the administrator can get a detailed view of parts of the farm without being forced to deal with the overall picture. Landscapes are created by specifying grouping criteria in the form of a SQL statement, which is applied to the Farm Management Information Database. There are two forms of landscapes, based on the actions that update the landscape grouping: If the data used is static or changes only infrequently, a trigger is added to landscape control components such that the landscape is recomputed whenever a field changes. The second form handles landscapes that provide views of data that is dynamic in nature, and where the landscape is recomputed on a periodic basis.

An example of the first form is grouping by physical location, by machine type or by assigned functionality. The second form generally handles grouping based information such as compute load, number of active transaction components, number of client connections serviced or available storage space, often using an additional level of grouping based on cluster functionality.

7 Cluster design and construction

Although the farm provides the administrator with a set of new tools to manage collections of nodes, they are too primitive to support the level of control, service provision and customization needed in the types of cluster computing one typically encounters in an enterprise data center. To support the specialized operation each cluster instance requires additional functionality, which must be added on a per-application basis to nodes that make up a cluster[1].

One of the most important tools in the farm management is the *Cluster Designer*, with which the administrator can group nodes together into a cluster and assign additional functionality in the form of *cluster service components*. Which components, and which component versions are needed for the operation of a particular cluster is specified in a predefined *cluster profile*, which is unique per type of cluster. The Cluster Designer manages these profiles, and a simple composition scheme is used to create new profiles for specialized clusters.

Basic in the operation of each cluster is a *cluster service*, which is a set of components running at each node in a cluster. The cluster service provides functionality similar to the farm service with which it cooperates in sharing network and state sharing resources. The cluster service augments the functionality of the farm service in the area the failure detection, which is a more focused version that exploits knowledge about the cluster to provide an as fast and reliable service as is possible, and in the area of cluster membership. The cluster membership is maintained using a consensus based membership protocol, guaranteeing that every member sees identical changes in the membership in the same order. The cluster service also provides additional communication services to the cluster components, and it provides component control and membership functionality similar to that of the farm service. The cluster service makes the cluster, the cluster member nodes and its cluster service components available to its members through a single unified naming scheme.

Using the Cluster Designer is often not sufficient to create a fully functional cluster. Frequently the cluster service components need additional configuration. For example the *web-clone cluster component* needs to be configured to find the source of the document tree to be replicated or which cyclic multicast file transfer service to subscribe to, to receive document updates. Other more complex configuration situations arise when the nodes are physically sharing resources such as storage area networks, which may need additional management actions. Even though Galaxy is able to provide a large part of the cluster management and support infrastructure, the administrator still will need to install application level functionality and provide the configuration of application components.

[1] In the metaphor of farms and landscapes, clusters can be visualized as *pastures*, and although we would have liked to use this term, we will continue to use cluster, as it contributes to a better understand of the work.

8 Cluster Communication Services

When a cluster-service instantiates the components it manages, it provides the components with an interface for intra-component communication and membership notification, similar to the service the farm-service provides to its components. The cluster-service communication facilities are different from those in the farm-service as they allow the cluster components to select communication properties such as reliability level and message ordering, or to provide flow and rate control parameterization. Cluster service components also have the possibility to create new communication channels within the cluster, to be used next to the basic intra-component channel.

The communication package used by the cluster-service is internally configurable, and the software components used inside the package are based on the particular cluster profile. For example if the cluster has access to multiple networks, or to a high-performance interconnect, it is the information in the cluster profile that determines which networks to use for administrative and intra-component communication, which networks can be used for additional communication created by the cluster service components, and which networks and techniques to use for failure detections.

9 Cluster profiles and examples

To support the initial Galaxy development goals we defined a number of cluster types and the cluster profiles that describe them. For two profiles we have developed a complete set of example cluster service components; the *Development Server* and *Component Server* are in daily use to support related research. The components of a third profile, the *Internet Game Server,* have been developed in the past months, and are currently under stress test. Of the other examples, the *Web-Clone* and the *Primary-Backup* profiles are making the transition from paper to real software design. An overview of the example cluster profiles is in table 1. In Figure 1 an example component layout is shown.

9.1 Application Development Cluster

This is a cluster type that is in daily use within the research group, and handles the functionality that is needed to support team development of cluster applications. The majority of the functionality of the components deals with easy integration into a development environment, with support for process & job control, install & version control, logging and debug support, and resource usage measurement and reporting. Process control, install services and debug message services have client parts that are integrated into Microsoft Visual Studio, providing a single point of access for development of cluster applications. Resource usage information is accessed at the developer side through a version of the Windows NT task manager that provides a cluster view instead of single node view.

Profile	Cluster service components
Application Development Cluster	Process & job control, install & versioning, debugging and distributed logging, Visual Studio integration, resource measurement.
Component Management Cluster	*System Services*: Component Runtime, Component & Component factory membership, Debugging, Logging and Monitoring, Isolation Service *Application Services*: State management, Global snapshots & Check-pointing, Synchronization, Consensus, Voting, Shared Data Structures
Game Server Cluster	Client management, Application level request routing, Specialized Measurement service, Synchronization services, State Sharing services, Shared VM service
Primary-Backup Cluster	Distributed Log Service, VM replication Service, Transaction service, Lock Manager, Fail-over & reconfiguration service
MSCS – Compatible Cluster	Fail-over Manager, Resource Manager &Controllers, Node Manager, Membership Manager, Event Processor, Database Manager, Object Manager, Global Update Manager, Checkpoint Manager, Log Manager
Cloned Service Cluster	Fail-over & reconfiguration service, Version & install service, Multicast file transfer service, Cache Control management.
Partitioned Service Cluster	Client Request Redirector, Distributed Log service, Synchronization service, Transaction service, Lock Manager, Shared State Manager, Fail-over & reconfiguration service
Parallel Computing Cluster	Job Queuing Manager, Process & Job Control, Synchronization service, Logging & Debugging service.

Table 1. Example Cluster Profiles

9.2 Component Management Cluster

One of the main objectives of our cluster area research is to develop a programming and execution environment for *Cluster-Aware* applications. This research is triggered by our past failures to provide useful high-level programming concepts for building advanced distributed systems. The services offered by the components in this cluster make up an execution runtime for cluster application implemented as components themselves, similar in concept to MTS or COM+/AppCenter [30].

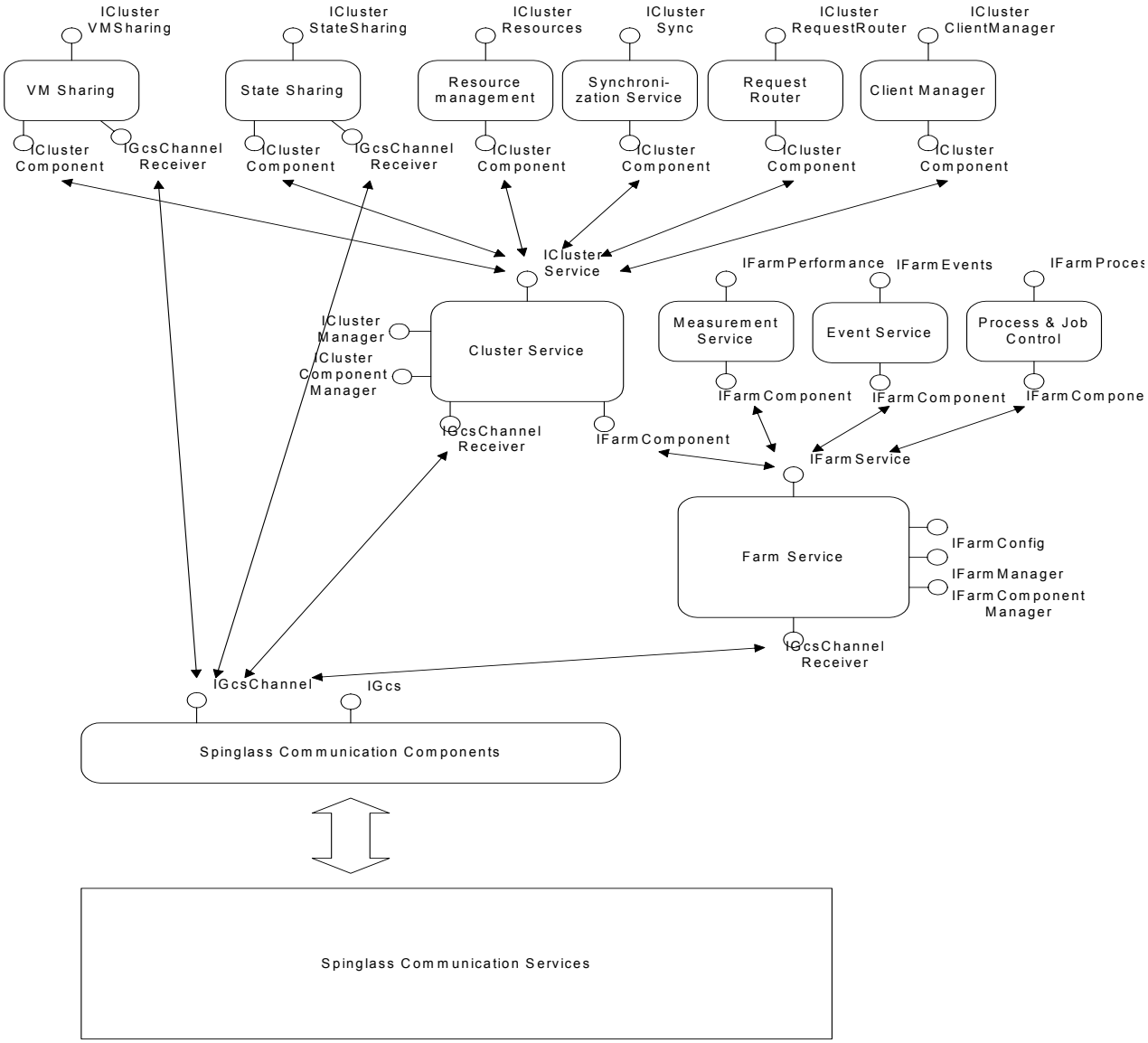

Figure 1. Layout of the various components in the Game Server Profile

The services are split into two categories: the first handles general management functionality: Component runtime management, which includes a functionality for providing component- & component factory membership, debugging, logging & monitoring services, and component isolation mechanisms for fault-containment. The second category handles the high-level distributed systems support for cluster-aware application construction: state management, global snapshots & check-pointing, synchronization, consensus, voting, shared data structures.

9.3 Game Server Cluster

One of the areas that provides us with the most complex technical challenges for providing scalable services is that of real-time multi-user Internet Game Servers. Clusters are an obvious match to provide a cost-effective "scale-out" solution, but the server side of the most games has not been designed with any cluster-awareness. In our research we are interested in investigating what support services are useful to provide on a cluster dedicated to serving Game Engines. Our first experiments involved restructuring one of the Quake server engines and build service components that allowed load-balancing as well as client fail-over. The initial results are promising, but the testing (simulating thousands of Quake clients) is still too difficult to report on solid results.

The cluster services that are developed to support the cluster version of the Quake game server are: client management, application level request routing, dedicated load measurement service, synchronization and state sharing services, and a shared VM service.

10 Component Development Libraries

Building farm and cluster components and their clients is made easier through the use of two libraries that we have developed. The first library supports the creation of the cluster components by providing classes that handle the interaction with the farm and cluster server, by parsing membership reports and incoming message types and providing those through event based processing structure. This library also supplies classes for handling clients from outside the cluster, in case this component supports direct connections made by client programs. It provides a scheduler centralizing the handling of input from the client, the cluster-services, and other component instances communicating through the cluster service, making it easier to building state machine structured distributed components.

The second library provides a collection of classes that makes it easier to build client applications that do not run on the clusters themselves. The classes encapsulate connection management such that a cluster name can be used at connection time, and the support classes manage the connection to the nodes in the named cluster, and automatically reconnecting on connection failure, possibly to another node in the cluster. There are also classes in both libraries that deal with forwarding membership information to clients, naming clients, and routing replies to client programs.

11 Related work

There is a significant body of work on cluster management systems, but none of the previous work provides a framework for farm management and supports for multiple styles as cluster computing in the way that Galaxy does. Most of the cluster management systems provide only support for very specific styles of management as they are built to support the operation of specific commercial clusters. Of these commercial clusters *VAXClusters* [14] and *Parallel Sysplex* [15] deserve special mention as they did groundbreaking work in building general distribution support for clusters. Microsoft's Cluster Service (MSCS) [27] has brought enterprise class clustering to the masses through a low-cost

solution for mainly fail-over scenarios. MSCS is limited in scale [28] and other styles of clustering are offered through separate Microsoft products such as NLB for web-site clustering or AppServer for middle-tier cluster management. In the Open Source community large number of packages are under development that support enterprise computing, of which two receive most attention: Linux-HA [23] which provides failure detection using a heart beat mechanism for Linux machines, and Compac's Cluster-Infrastructure for Linux [7] which provides a re-implementation of some of the Tandem NonStop Cluster membership and inter-node communication technology for Linux machines.

Cluster management systems supporting compute clusters are ubiquitous, most of them built at the major research labs to support custom clusters constructed over the past decade, e.g. [9]. With the advent of a trend towards more cost effective parallel computing, the *Beowulf* [25] package for parallel computing under Linux has become very popular, supporting a large number of compute cluster over the world. New directions in parallel computing require a more fine-grained, component based approach, as found in NCSA's Symera [10].

Historically there were only a few high-profile management systems for high-available cluster computing, which often was supported by dedicated cluster hardware support such as the systems by Tandem and Stratus [13]. Since the move to more cost enterprise systems most of the vendors have developed commercial cluster management products for high availability. All of these systems have limited scalability [28] and are very specific in terms of hardware support or supported configurations. An overview of the various vendor products can be found in [17].

Scalability in cluster systems has been addresses in cluster research, but mostly from an application specific viewpoint: The *Inktomi* [11] technology is a highly scalable cluster targeted towards Internet search engines and document retrieval. A cluster based SS7 call processing center [12] with good scalability from a real-time perspective, was build in 1998 using Cornell communication technology. Another cluster system that claims excellent scalability is *Porcupine*, which is very specifically targeted towards Internet mail processing systems [24].

In terms of specific distributed systems support for cluster management systems, there is only limited published work. Next to the systems based on Cornell research technology, the *Phoenix* system from IBM research has similar properties and is also applied to cluster management [1]. In the *Oceano* project research at IBM have started to experiment with automatic configuration and management of large datacenters, the results of which are fueling the *Autonomic* computing projects that focuses on automatic management and self-healing systems.

12 Evaluating Scalability

One of our activities that has not received much attention in this paper but is a very important part of our research is ongoing work on developing a systematic approach to evaluating scalability. In this paper we have made a number of claims about the scalability of the Galaxy farm and cluster components based on our theoretical and experimental results. It is outside of the scope of this paper to present the detailed experimental and usage results and we refer the interested reader to [5,6,16,19,21,22] for performance and scalability details..

There are many aspects to the scalability of distribution services. More research is needed before we are able to identify, isolate and measure, in a scientific manner, the scalability of systems. Providing a framework for reasoning about the scalability of complex distributed systems such as Galaxy, is one of our highest research priorities.

13 Acknowledgements

First and foremost our thanks go to Jim Gray for his continuous support over the years of this research. Without his advice, insights and intellectual stimulation the Galaxy architecture would never have risen to meet real world needs.

The large group of students who have implemented parts of Galaxy over the years deserve special credit, especially Dan Dumitriu who was responsible for the implementation of a significant part of the distributed systems technology used in Galaxy.

References

[1] Badovinatz, P., Chandra, T.D., Gopal, A., Jurgensen, D., Kirby, T., Krishnamur, S., and Pershing, J., "GroupServices: infrastructure for highly available, clustered computing", unpublished document, December 1997

[2] Birman, K.P, "The Process Group Approach to Reliable Distributed Computing", *Communications of the ACM*, vol. 36, no. 12, December 1993

[3] Birman, K.P., and Renesse, R. van, "Software for Reliable Networks", in *Scientific American*, May, 1996

[4] Birman, K.P., *Building Secure and Reliable Network Applications*. Manning Publishing Company, and Prentice Hall, 1997

[5] Birman, K.P., "A Review of Experiences with Reliable Multicast". *Software Practice and Experience*, vol. 9, 1999

[6] Birman, K.P, Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M. and Minsky, Y., "Bimodal Multicast". *ACM Transactions on Computer Systems,* vol. 17, no. 2, May 1999

[7] Cluster Infrastructure for Linux, http://ci-linux.sourcefourge.net, December, 2001.

[8] Devlin, B.; Gray, J.; Laing, B.; Spix, G., "Scalability Terminology: Farms, Clones, Partitions, and Packs: RACS and RAPS", *Microsoft Research Technical Report,* December, 1999

[9] Fineberg, S.A., and Mehra, P., "The Record Breaking Terabyte Sort on a Compaq Cluster", in the *Proceedings of the 3rd Usenix Windows NT Symposium*, Seattle, WA, July 1999.

[10] Flanigan, P. and Jawed Karim J., "NCSA Symera Distributed parallel-processing using DCOM", *Dr. Dobb's Journal,* November 1998

[11] Fox, A., Gribble, S.D., Chawathe, Y., Brewer, E.A., Gauthier, P., "Cluster-Based Scalable Network Services", *in Proceeding of the 16th ACM Symposium on Operating Systems Principles*, Sant Malo, France, September 1997.

[12] Friedman R., and Birman, K.P., "Using Group Communication Technology to Implement a Reliable and Scalable Distributed IN Coprocessor", in the *Proceedings of TINA 96*, Heidelberg, Germany, 1996.

[13] Horst, R.W., and Chou, T.C.K., "An Architecture for High-Volume Transaction Processing", in *Proceedings of the 12th International Symposium on Computer Architecture*, Boston, MA, 1995.

[14] Kronenberg, N., Levy, H., and Strecker, W.,"VAXclusters: A Closely Coupled Distributed System," *ACM Transactions on Computer Systems*, vol. 4, no. 2 (May 1986)

[15] Nick, J.M., Moore, B.B., Chung, J.-Y., and Bowen, N., "S/390 cluster technology: Parallel Sysplex", *IBM Systems Journal*, Vol32, No. 2, 1997.

[16] Ozkasap, O., Renesse, R. van, Birman, K.P., and Zhen Xiao, Z., "Efficient Buffering in Reliable Multicast Protocols". *in Proceedings. of NGC99*. Pisa, Italy, November 1999

[17] Pfister, G.F.,*In Search of Clusters : The Ongoing Battle in Lowly Parallel Computing*, Prentice Hall, 1998.

[18] Ranganathan, S., George, A., Todd, R., and Chidester, M., "Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters", *Cluster Computing*, Vol. 4, No. 3, July 2001

[19] Renesse, R. van, Yaron Minsky, Y., and Hayden, M., "A Gossip-Based Failure Detection Service", in *Proceedings. of Middleware '98*, Lancaster, England, September 1998.

[20] Renesse, R. van, Birman, K., Hayden, M., Vaysburd, A., and Karr, D., "Building Adaptive Systems Using Ensemble", *Software--Practice and Experience*, August 1998.

[21] Renesse, R.van,. "Scalable and Secure Resource Location"*, In Proc. Of the 33rd Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2000

[22] Renesse, R. van, and Birman, K., "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management and Data Mining", submitted to ACM Transactions on Computer Systems (TOCS), November 2001.

[23] Robertson, A., "Linux-HA Heartbeat System Design", *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GE, October, 2000.

[24] Saito, Y., Bershad, B.N., and Levy, H.M, "Manageability, Availability and Performance in Porcupine: A Highly Scalable, Cluster-based Mail Service", *in Proceeding of the 16th ACM Symposium on Operating Systems Principles*, Charleston, December 1999

[25] Sterling, T., et al., *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters,* MIT press, June 1999.

[26] Vogels, W., "World Wide Failures", *Proceedings of the 7th ACM SIGOPS European Workshop*, Conamora, Ireland, September 1996

[27] Vogels, W., Dumitriu, D., Birman, K. Gamache, R., Short, R., Vert, J., Massa, M., Barrera, J., and Gray, J., "The Design and Architecture of the Microsoft Cluster Service -- A Practical Approach to High-Availability and Scalability", *Proceedings of the 28th symposium on Fault-Tolerant Computing,* Munich, Germany, June 1998.

[28] Vogels, W., Dumitriu, D., Agrawal, A., Chia, T., and Guo K., "Scalability of the Microsoft Cluster Service", *Proceedings of the Second Usenix Windows NT Symposium*, Seattle, WA, August 1998.

[29] Vogels, W., van Renesse, R., and Birman, K., "Six Misconceptions about Reliable Distributed Computing", *Proceedings of 5th SIGOPS European Workshop*, Sintra, Portugal, September 1998.

[30] Vogels, W., Chipalowsky, K., Dumitriu, D., Panitz, M., Pettis, J., Woodward, J., "Quintet, tools for building reliable distributed components", *Proceedings of the 2nd International Enterprise Distributed Object Computing Workshop,* San Diego, November, 1998.