Application Communication Infrastructure: A Radical Approach

Introduction

A system of multiple computers can and should work together to survive the failure of one computer or all computers at one location. A failure is due to an adverse event such as a successful Cyber-attack, power failure, fire, flood, or any catastrophic event. Work being performed by a failed computer or all computers at one location can and should be quickly recovered and continued after a failure.

A solution must be more than multiple processors doing the same work at the same time where a mission critical system survives a processor failure. Work must shift to an alternative location when a location is lost due to a catastrophic event.

Applications on multiple computers use an infrastructure to communicate data via network hardware and software. A communication infrastructure, for a system to continue work after a failure, should also:

- Minimize effort to create applications that distribute, isolate, and recover work.
- Provide reasonable cost and performance.
- Enable applications to converse peer to peer, client to server, or publisher to subscriber.
- Encrypt application conversations.
- Enable a client to server conversation to be recoverable.
- Help to ensure and prove software corruption cannot spread.
- Enable an administrator to control and audit data flowing between applications.
- Enable change during operation by supporting multiple simultaneous versions and ensuring conversing parties are the same version.

An existing communication infrastructure likely does not and almost certainly cannot achieve all these goals. An existing infrastructure is based on conventional wisdom. Conventional wisdom comes from what is possible for one computer alone or two computers working together. A system and infrastructure achieving these goals must use many more than two computers to survive the loss of a location. Therefore, an infrastructure must be based on unconventional or radical approaches.

A communication infrastructure must provide features that fulfill application needs such as:

- Data is represented in a format as needed by a computer, software language, and person.
- Two or more applications can converse via messages containing data.
- Applications can converse as a client requesting a function of a remote server.

Data Representation

No single data format satisfies all needs. A person communicates data as a language in a text format. A programmer and software language typically represent data as a class. A computer needs to process data as binary zeros and ones. Computers exchange data as serial (zeros and ones) or text.

The need to represent the same data in different formats creates a requirement to convert data between formats. There are many solution approaches for data conversion, often with a data format syntax chosen for easier conversion. Each approach works but does not meet one or more of the stated requirements. A full analysis of approaches and unmet requirements is not useful for the purpose of this paper.

A radical solution approach is for an infrastructure to have only one universal format conversion algorithm to convert any data content between formats. A universal converter is made possible by requiring a programmer to specify meta data that directs a conversion of data one primitive data attribute at a time. A software language can efficiently perform this kind of conversion given a few extra hints. Unfortunately, the cost of implementation and need for software language committee approvals is too great for this solution approach to be viable for the immediate future.

An alternative approach is to define an infrastructure class per primitive data type to hold a data value and conversion hints. An application class is defined with underlying infrastructure classes to hold a data value and conversion meta data. An application initializes meta data as part of a new class instance. A conversion uses this meta data to convert any data content between formats one data value at a time. This approach uses more programmer effort and local memory than a software language approach but is available immediately. Most importantly it meets the goal of minimizing application effort by eliminating the programing needed for a conversion between formats of each unique type of data content.

Message Communication

An application sends a message to another application via an infrastructure. An infrastructure sends a message from one computer to another computer via network hardware and software. An administrator needs to control the flow of data between computers so that a message is only originated from, delivered to, and understood by an authorized computer.

A conventional solution approach requires application to application connections and a unique encryption key per connection to properly secure data communication. This approach requires on the order of "(n – 1) squared" application to application connections and keys where "n" is the number of applications in a system.

A hub network is a more efficient but less frequently used solution approach. A hub can enforce where a message can originate from and where it goes. A unique key per application is used to encrypt a message when in transit between an application and hub. This approach requires "n" application to hub connections and keys where "n" is the number of applications in a system.

Combining a hub network together with a local hub within a computer, call it a local and central hub network, is an even more efficient but less frequently used solution approach. No key or encryption is needed when a message transits inside a computer between an application and local hub. A local hub creates a network connection to a central hub network and multiplexes local application instance messages onto the network connection. This approach requires "m" application to local hub connections where "m" is the number of application instances on one computer. The approach also requires "n" local hub to central hub connections and keys where "n" is the number of computers in a system.

A radical solution approach is a local and redundant central hub network. There are two independent hub networks. A local hub creates a network connection to two central hubs. This approach requires "m" application to local hub connections where "m" is the number of application instances on one computer. This approach requires "2 * n" local hub to central hub connections and "n" keys where "n" is the number of computers in a system. This is a variation of sending a critical message via two paths.

An application calls an application resident infrastructure to send or receive a message. The application resident infrastructure connects to a local hub on the same computer. A local hub connects to two hub networks and sends a message to both hubs. A receiving local hub sends the first arriving message copy to an application resident infrastructure and discards any subsequent copy.

Application Communication Infrastructure: A Radical Approach

An application sends and receives a message as content within a data class. An application resident infrastructure converts a message between an application class and transmitted serial. An administrator can configure that a message is converted to a text format and added to an application log file.

A redundant application & hub network is an ideal solution approach for a published message to be efficiently distributed to subscribers. A unique identifier is assigned for each type of published event message. Each publisher has only one active instance at a time so that a subscriber instance either receives each event message published for a subscribed event type or is terminated if a delivery fails. A starting subscriber is automatically given the last published message for each subscribed event type. An active and standby subscriber receive published messages in the sequence of publication.

A local hub sends a published message to each central hub. A central hub forwards it to a computer with subscribers as specified by an administrator. A destination local hub distributes it to local application subscribers. Multiple events are multiplexed onto a connection used only for events to ensure efficient and timely event distribution.

Some advantages of a redundant application & hub network are:

- Avoid an intermittently failed or slow network.
- Flow control messages to avoid exceeding network capacity and discarding a message.

Client to Server Communication

An application can act as a client to request a function from a remote server. A conventional client to server solution approach directly connects a client and server.

A radical solution approach is to insert a manager between a client and server. A client and server make a connection to a manager via the message communication concept. A client uses a session to communicate with a remote server. Recovery information is placed in a session and included with each function request. A manager receives a request and recovery data, stores both, replicates both to a backup manager, and forwards a request to a destination. If a client instance fails, then a manager assigns a session for recovery by a backup client instance. The manager makes the backup client appear to be the original to the server. The server is only aware of and affected by the time needed for failure detection and recovery. A server similarly uses a session with recovery information that is passed to a manager to recover a server session after a failure.

Work Distribution Layer

The radical approach of data representation, message communication, client to server communication, and other features are packaged together as a work distribution layer. An application can act as a peer, client, server, or any combination of all three.

There are many advantages to the work distribution layer such as:

- Minimize effort to write client and server software working together as one system.
- The conversion of a message between class format and serial by the infrastructure means errors such as buffer overrun, illegal text insertion, and a mismatch of message content between sender and receiver are avoided or detected and rejected by the infrastructure. This helps ensure and prove software corruption cannot spread to another system computer.
- The state of clients and servers are monitored so that a client connects to an active server.

- A starting instance can connect to an active instance of the same program to synchronize data.
- Configuration and software changes are updateable one computer at a time by supporting multiple versions to execute at the same time. A change mechanism ensures that a client instance and a server instance are at the same version.
- Client and server identity are established via use of an encryption key.
- A single system authentication server can establish user identity.
- An administrator controls what a user and client can request from a server.
- A session used for a client to server communication allows one application instance to handle multiple users. User messages are multiplexed onto one connection. The infrastructure can switch between user sessions as needed to handle an incoming message. Work for more users can be performed given the same amount of computer and network resources.
- Maximize work reliability, security, scalability, and performance at a reasonable cost.
- Testing of applications is possible on one computer by combining local and central hub functions into a special test communication infrastructure.

Service Provided by Server for Client Use

A server provides a client-side service to minimize the effort needed to create a client that uses a server. A service offers an API of client callable methods. A method maps each method parameter to and from server request and response messages held in infrastructure data conversion classes. Contents of a request class are converted to serial and sent to a manager. A manager restricts a client to only communicate with servers per administrator configuration. A manager informs a server what requests a client can make. A server receives a request in serial that is converted back into a class format. A server validates that a request is allowed before preforming work. A client response message is handled in a similar fashion.

A test client can validate that a server detects and prevents any unauthorized client request.

Summary

Any system is less costly and more reliable and secure by becoming a distributed system with internal error detection, in-progress work recovery upon a failure, and avoidance of network congestion or failure. An application uses a server supplied service to request server work at minimal cost. A server supplied service exchanges a message with a server via the work distribution layer. This software pattern guides the development of better software with less effort.

An administrator easily controls where client and server applications reside, what requests a client can make of a server, and whether a message simultaneously travels two network paths.

A communication infrastructure can combine these solution approaches with several others for use by an application. Visit web site SoftEcoSDK.com or send an email to SoftEcoSDK@gmail.com for more information. Research and development are needed to transform a proof of concept implementation of mostly production quality mechanisms into a commercial product.