Architecting high-performance data-centric systems using data distribution service and switched fabrics

Many modern large-scale applications can be characterized by three attributes: they need to gather and distribute data in real-time, the amount of data being transferred is significant, and the entities involved in this data exchange are varied and may even change over time. In this article, the author shows that you can improve the performance characteristics of your design if you combine switched fabrics and data distribution service-compliant middleware to give designers a tunable application environment that will meet the needs of disaggregated applications spread across multiple networked nodes.

By Rajive Joshi

Many modern large-scale applications can be characterized by three attributes: they need to gather and distribute data in real-time, the amount of data being transferred is significant, and the entities involved in this data exchange are varied and may even change over time. For instance, air traffic control, financial transaction processing, battlefield or naval command and control, or industrial automation systems all are examples of data-centric systems.

The data-centric architecture flattens information or data distribution patterns, making data sources directly available to any authorized node on the network that wishes to use the data. Applications, once centralized on high-performance servers, can now be disaggregated; applications will span multiple, networked computers. Disaggregated or distributed systems are much easier to scale, make fault-tolerant, and upgrade. This distribution calls for new application infrastructures.

Publish-subscribe communications is a key enabler of the new distributed architecture. Data sources publish their data to the network; data users subscribe to the data to receive real-time updates. This data-centric transformation is revolutionizing embedded and real-time system design as well.

To disaggregate or distribute real-time embedded systems, new technologies are needed. On the hardware side, switched-fabric networks serve as high-performance communications conduits. To support distributed real-time software applications, new real-time publish-subscribe network software or middleware provides the data distribution services needed. The new data distribution service (DDS) for real-time systems standard from OMG defines a publish-subscribe mechanism that maps naturally to the topologies and capabilities of switched fabrics (Figure 1).

An ideal system platform

Developers of today’s distributed applications find their system requirements are becoming increasingly more challenging. Today’s architectures are typically based on a multi-CPU, VME-backplane solution with hard-wired input/output interfaces to whatever sensors and effectors the system needs to do its job. Other systems that make up the entire application are considered late in the development cycle, if at all. The capability, supportability and availability of the entire application are not really a consideration for the subsystem developer.

What is an ideal solution? Would be nice, but not essential, if the multi-CPU, backplane shared-memory architecture could just be scaled up infinitely. The backplane would have zero-latency and infinite bandwidth. The quality-of-service (QoS) would be perfectly matched to the latency and bandwidth requirements of each data stream. Each application would be a separate operating system process and communication with other processes would be through a common, well-specified messaging interface. The software components would be dynamically distributed anywhere in the system to the location where its execution is optimized by the underlying hardware as opposed to where it must reside in order to satisfy inflexible architectural requirements of fixed software interfaces. Adding new functionality, system upgrades, and fault-
tolerance would be straightforward with an ideal architecture like this.

DDS and switched fabrics

While not ideal, switched fabrics combined with data distribution service (DDS) get us much closer. Designers of complex, distributed systems stand to gain from the emergence of switched fabrics and the DDS standard. Table 1 shows some of the common requirements that these designers are faced with and how those requirements are typically met with existing bus backplane technologies. The complexity of software needed to implement current solutions to these common requirements increase system development and post-deployment maintenance costs. In some cases, the required complexity results in projects that fail to complete development.

Switched fabrics such as StarFabric, PCI Express AS and Serial Rapid IO give new freedoms to designers to implement their systems (Figure 2). A switched-fabric bus is unique in that it allows all nodes on the bus to “logically” interconnect with all other nodes on the bus. Each node is physically connected to one or more switches. Switches may be connected to each other. This topology results in a redundant network or “fabric,” in which there may be one or more redundant physical paths between any two nodes. A node may be logically connected to any other node via the switch(es). A logical path is temporary and can be reconfigured or switched among the available physical connections. Switched fabric networks can be used to provide fault tolerance and scalability without unpredictable degradation of performance, among other features.

DDS excels at real-time data distribution. DDS data-centric publish-subscribe is characterized by a set of data producers and data consumers (Figure 3) that communicate over common named “topics.” A topic is identified by a name and a data type. A data producer declares the intent to publish data on a topic; a data consumer registers its interest in receiving data published on a topic. The middleware acts as the glue between the producers and the consumers; it delivers the data published on a topic by a producer

Table 1. New applications demand increasing capability, supportability and availability. Yesterday’s centralized, stove-piped system architectures don’t respond well.
Figure 2. Switches integrated into bridge devices provides for fabric path redundancy without using up a chassis slot for a separate switch. External switches can be added to expand the fabric and provide additional redundant paths.

Figure 3. DDS publish-subscribe involves direct anonymous communication between producers and consumers of data. Topic A has a primary producer 1, and a backup producer 2. Note that nominally when producer 1 is active consumers do not receive data from the backup producer 2.

The power of QoS parameters

Publish-subscribe is a powerful paradigm. But the real key to DDS’s power is its ability to flexibly, but precisely, specify performance requirements between all the different parts of the system. DDS achieves this power through the pervasive use of QoS parameters. QoS parameters configure the system and establish contracts between publishers and subscribers that specify exactly how information should flow between those nodes. QoS contracts provide the performance predictability and resource control required by real-time systems, while preserving the modularity, scalability and robustness inherent to the anonymous publish-subscribe model.

DDS allows for fine-grain control over QoS on a per-data-flow basis. This aspect, unique to DDS, enables application designs that easily support extremely complex designs, but with flexible data flow requirements. QoS parameters control virtually every aspect of the DDS model and the underlying communications mechanisms. The DDS middleware is responsible for ensuring that participants meet the level-of-service contracts guarantees the predictability of operations that is necessary for real-time systems.

Table 2. Examples of DDS implementation-specific QoS policies that enable tuning the software implementation to the hardware needs.

<table>
<thead>
<tr>
<th>QoS Policy Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_WRITER_PROTOCOL</td>
<td>The rate at which the reliable protocol sends heartbeats; and when it switches between different modes, allows an application to obtain lower latencies and faster recoveries in the presence of packet loss at the expense of additional bandwidth and CPU consumed.</td>
</tr>
<tr>
<td>DISCOVERY_CONFIG</td>
<td>The rate at which each participant re-asserts its presence in the network allows an application to control how fast it will be notified and respond to computer or network failure at the expense of additional CPU and bandwidth.</td>
</tr>
<tr>
<td>HISTORY</td>
<td>The size of the sending queues allows an application to control how much bandwidth a writer can use when running on top of networks with long round-trip delays at the expense of longer recovery times in case packets are dropped.</td>
</tr>
<tr>
<td>DATABASE</td>
<td>The sizes of the hash tables used internally allow the application to configure how well performance scales when the number of topics increases at the expense of memory used.</td>
</tr>
<tr>
<td>DOMAIN_PARTICIPANTRESOURCE_LIMITS</td>
<td>The ability to size initial and maximal buffers allow an application to control when and how much memory is consumed to achieve predictable behavior, at the cost of having to model the whole system better and requiring re-configuration of the system if it were to expand beyond its initial design.</td>
</tr>
<tr>
<td>EVENT</td>
<td>The control of the priorities of the internal threads gives the user the ability to place them at an optimal place with respect to those of other processes in the system at the expense of having to understand and model the whole system better.</td>
</tr>
<tr>
<td>TRANSPORT_SELECTION</td>
<td>The ability to use dedicated ports, transports, and addresses for each DataReader allows a developer to separate the resources used for high-priority traffic versus those used by other traffic at the expense of consuming more kernel resources (sockets, mutexes, threads, memory, etc.).</td>
</tr>
</tbody>
</table>
Tunability means optimal performance

Real-Time Innovations (RTI), a co-author of the DDS OMG specification, provides a DDS-compliant implementation, called the “RTI Data Distribution Service.” RTI has gone beyond the DDS standard to offer features that allow designers to tune their application while still complying with the DDS standard. Tunability is about the ability to marry the performance characteristics of both the hardware and the software, sometimes dynamically, to changing distributed system requirements, trading off performance needs on the network with a whole gamut of other issues, such as memory use, local CPU loading or reliability.

New QoS policies

RTI accomplishes this by adding a number of optional QoS parameters (Table 2) to the RTI data distribution service, to allow a developer to optimize the implementation under different use cases. These parameters are tunable independently for each publisher or subscriber so that an application can tune in each reader and each writer for the optimal combination of resource use and performance.

This tunability brings significant benefits in terms of flexibility and scalability. Disaggregated applications don’t have to make fixed or rigid decisions on resource allocation, but can tune themselves dynamically for optimal performance. And as the entire system grows and changes, the application can make performance or resource allocation decisions as often as necessary.

Conclusion

You can improve the performance characteristics of your design if you combine switched fabrics and DDS-compliant middleware like RTI. When married with switched fabrics, it provides a tunable flexible application environment that will meet the needs of disaggregated applications spread across multiple networked nodes.

References


ABOUT THE AUTHOR

Rajive Joshi is a principal software engineer at Real-Time Innovations Inc. He specializes in the design of distributed and real-time systems, emphasizing object-oriented and component-based software architecture. He has served as a consultant and developer for distributed systems projects in the areas of robotics and automation, including Alstom Schilling’s Quest remotely operated vehicle and CrouseHinds’s automated filament-alignment system.


He holds a PhD and an MS in Computer and Systems Engineering from Rensselaer Polytechnic Institute, and a Bachelor of Technology in Electrical Engineering from the Indian Institute of Technology (IIT) at Kanpur, India. He is a member of the IEEE, ACM, and AIAA. He can be reached at Rajive.Joshi@rti.com.