2.3 ORPG REVIEW: CLUSTERING, DATABASE AND SYSTEM MANAGEMENT

by Zhongqi Jing[1]*, Steve Smith[2] and Michael Jain[3]

[1]Cooperative Institute for Mesoscale Meteorological Studies,
University of Oklahoma, Norman, Oklahoma

[2]NOAA/NWS/WSR-88D Radar Operational Center

[3]NOAA/ERL/National Severe Storms Laboratory

1. INTRODUCTION

Several ORPG (the Open systems Radar Product Generator) design concepts have been discussed in our previous IIPS conference presentations (Jain et al., 1977, Jing et al., 1997, Jing et al., 1998, Jing et al., 2000). In this paper we will present several others, namely the clustering technology, the embedded data query support and the ORPG management and monitoring.

Clustering is a widely used architecture to implement scalable, highly-available systems. The ORPG clustering technology is based on a loosely-coupled multi-tasking software architecture. It allows the ORPG system to run on multiple nodes (UNIX hosts, e.g. workstations and PCs) and thus, makes ORPG scalable. When the processing resource requirement increases beyond the current RPG hardware limitation, new hardware such as PCs can be incrementally added to the system to scale up the processing capability.

ORPG has a built-in highly efficient and light-weight data query system for managing products and product users. The technology, an embedded simple database system, will be discussed and compared with other database systems.

The ORPG is a loosely-coupled multi-tasking system. It consists of nearly one hundred processes and as many data stores which are being updated dynamically for publishing and exchanging data. We will discuss system management issues such as system start-up and shutdown, process monitoring and system status tracking.

Understanding the technology and design concepts of the ORPG system should be beneficial for its users, developers and maintainers. The ORPG technology and the software could potentially be reused by other similar systems if awareness of its capabilities are increased.

* Also affiliated with NOAA/OAR/National Severe Storms Laboratory. *Corresponding Author address*: Zhongqi Jing, NSSL, 1313 Halley Circle, Norman, OK 73069; e-mail: jing@nssl.noaa.gov.

2. CLUSTERING

One of the ORPG design goals is to support increasingly more data processing applications. As new algorithms and new products are introduced to the system, the demand on the system resources (CPU, memory, disk storage, network bandwidth and so on) will increase. Clustering is a widely used architecture to implement scalable, highly-reliable systems. It allows incremental addition of new hardware to provide required resources. The ORPG supports network based clustering, in which multiple, general purpose UNIX workstations and/or LINUX PCs (called nodes) are connected through a TCP/IP network. No special clustering hardware is necessary. The ORPG software then runs on all available nodes simultaneously. The ORPG clustering feature is sometimes called distributed processing.

Clustering makes ORPG scalable. When the processing resource requirement increases beyond the current RPG hardware limitation, new hardware such as PCs can be incrementally added to the system to scale up the processing capability. Compared to replacing the entire RPG hardware, this will reduce hardware upgrade risk and allow RPG processing power to be scalable virtually without limitation.

RPG performance and availability can be increased through clustering. Should a workstation fail, it is possible to run RPG (perhaps in a degraded mode if the resources are not sufficient) without the failed hardware. Clustering also provides the potential for dynamic load balancing, which can further improve RPG performance and availability. Dynamic load balancing is currently not implemented.

The ORPG clustering technology is based on the ORPG loosely-coupled, distributed, multi-tasking architecture (Jing et al., 2000 and Jain et al., 1997) which features the following concepts:

1. ORPG software consists of a number of tasks (applications). New tasks (e.g. tasks for implementing new algorithms, generating new products and providing new infrastructure services), can be added.

2. ORPG tasks communicate with each other through data stores. Although other inter-process communication (IPC) mechanisms, such as remote procedure call (RPC) and

client-server model, are supported in ORPG, communicating through data stores is the primary means of IPC. This makes the RPG a loosely coupled multi-tasking system.

3. ORPG tasks are built on top of the ORPG infrastructure libraries, which make the clustering possible. The ORPG infrastructure software provides support for reliable data access regardless of the physical location of the data store and the node characteristic (e.g. data byte order). In addition to data store access, the ORPG infrastructure software allows the applications to call any function and execute any command on remote nodes.

The ORPG clustering support is mostly transparent to the application developer and the system user. The following are some of the design decisions.

1. ORPG applications will be able to run in the distributed environment without any special coding. That is, any application developed on a single-node environment will be ready for running in a multi-node configuration.

2. All nodes are identically installed. i.e. All hosts are loaded with the same libraries, executables and ORPG configuration files. There is no master node designation.

3. A system configuration file is generated and distributed to all nodes when ORPG is started. This file contains information about locations of all ORPG public data stores. The infrastructure software uses this file to access the appropriate data stores which contain all ORPG internal data and meteorological data. The system configuration file can be updated while the ORPG is running. This will be useful for dynamic load balancing.

4. Each application creates and writes its log file on its local node. The tool that reads and prints the log messages (called “lem”) is able to run on any node and transparently access the log file of any task(s).

5. The status of all operational processes (tasks) is monitored regardless of their location. The tool that prints the process status (called “rpg_ps”) can be invoked on any of the nodes.

6. The human computer interface (“hci”) can run on any of the nodes.

7. Command line ORPG control commands, such as startup, shutdown and others, can be issued on any of the nodes.

The ORPG uses a configuration file for specifying how to distribute its data stores and operational processes among available nodes. If this file does not exist, ORPG will run on the local node assuming a single node configuration. Multiple configurations may be prepared for load balancing and dealing with hardware failure. Each file can contain multiple configurations.

Listed in the following is a simple example resource configuration file for illustrating how to use the ORPG clustering feature.

```
node_names {
        rpg         apple
        pc          pear
}
all_nodes {
        rpg         ras_sharedBasedata   ras_default
        pc          ras_baseProducts     ras_sharedBasedata
        replicate   55          rpg         pc
}
single_node {
        rpg
}
ras_baseProducts {
        product 2 3 4 5 6 7
        task basrflct
        product 11 12 13 14 15 16
        task basvlcty
}
ras_sharedBasedata {
        product 55                      # BASEDATA
}
```

In this particular cluster configuration file, we have two nodes. We assign a name to each of the nodes. In this case, the node names are "rpg" and "pc". Node "rpg" is a SUN workstation which, for example, could be the RPG host in the current WSR88D operational system. Node "pc" is a PC running LINUX which could be an additional processor to the operational system. In the first section (with key word "node_names"), we specify the host name (or IP address) for each node. In this case, the host names are "apple" and "pear" respectively.

We define two clustering configurations in this file. They are named with "all_nodes" and "single_node". Configuration "all_nodes" uses all nodes (two in this case) and "single_node" uses only node "rpg".

In configuration "all_nodes", we assign resource sections "ras_sharedBasedata" and "ras_default" to node "rpg" as seen in the file. Similarly, we assign resource sections "ras_baseProducts" and "ras_sharedBasedata" to node "pc". Each named resource section, which must be defined in the file, specifies a set of processes and data stores. Resource section named "ras_default" does not need to be defined. It represents all data stores and tasks that are not explicitly allocated to any node. We can see, in this example, we allocate the two tasks, "basrflct" and "basvlcty", to node "pc". Data stores, "2" through "7" and "11" through "16", are allocated on node "pc". These are the radar base products. All other tasks and data stores are allocated on "rpg".

Thus, in this configuration, the base products are generated on node "pc". In order to generate the base products, task "basrflct" and "basvlcty" will need the base data which, in this example, are accessed from data store "55". If we only allocate "55" on "rpg", both tasks will have to access "55" individually (across the network in this case). To reduce network traffic, we duplicate data store "55" on both nodes and invoke the data replication service (line "replicate 55 rpg pc" in the file). This allows all of the tasks running on the PC to locally access the radar base data.

In the "single_node" configuration, we do not need to specify any details because everything will be on the default nodes which is the first node if not otherwise specified.

3. DATA QUERY

ORPG has a built-in, highly efficient and light-weight data query system for managing products and product users. The technology, an embedded, simple database system, adds content based data search support to the ORPG.

Almost all data in the ORPG, including base radar data, products, intermediate products, communication messages, adaptation data, configuration data, status and state data and so on, are stored in data store objects called Linear Buffers (LB) (Jing et al., 2000). A Linear Buffer is a managed file or shared memory segment that can store multiple messages. Data stored in an LB are accessed through the LB library routines. The LB library provides support for data persistence and integrity, event notification, remote access and concurrence access control. Data (messages) in an LB are organized as either a message queue or a message pool. Messages can be accessed sequentially or randomly with message IDs. Although the LB supports certain data base features, by itself, it does not support content based access such as data query.

The ORPG data query technology adds data query support for messages stored in LBs. The client-server architecture is used here. A server is implemented to provide the query service. The server retrieves and maintains information that is required for query service from all messages of the LB supported by the server. It updates the message information when any message in the LB is updated, deleted or inserted. When any application in the ORPG needs to query the LB, it sends a request (through an SQL-like API described later) to the server. The server performs the search and returns the result to the client. The client application can be on any host on the network.

The client server model eliminates the need for each individual application to maintain a copy of the message information and follow message updates in the LB. Each instance of the server provides query support for a set of any number of specified LBs. Multiple instances of the server can be started for different sets of LBs. The LBs supported by the server can be on different hosts.

The server maintains the message records in a table linked to “Red-Black” (balanced binary) trees called index trees. The index trees allow efficient (logarithmic complexity) record insertion, deletion and search. The efficiency is important in real-time data processing systems with large amount data to manage. The ORPG data query technology supports heterogeneous environment (e.g. one consists of both PCs and SUN workstations).

The ORPG data query technology uses a simple API resembling the SQL's "where" statement. For example, the following function call,

SDQ_select ("product.lb", "vol_num = 34 and prod_id = 19", &result); searches for all products of product ID "19" and volume number "34" stored in the product LB named "product.lb".

The second argument in calling SDQ_select is the "where" statement similar to that used in the SQL query of a typical relational data base. For example, the following "where" are acceptable: "vol_num < 13 and vol_num > 10 and (prod_id = 19 or prod_id = 20) and not elevation > 5".

In the current ORPG, products and product users are accessed through the query interface. The data query feature can be used in the future in other areas such as adaptation data and algorithm development.

An illustrative example is querying an adaptation data LB:

SDQ_select ("RPG_adapt", "radar_name = KOUN and data_name = 'radar location'", &result);

This call could return a structure containing the latitude, longitude and altitude of radar "KOUN".

Algorithms could use the data query service for managing their internal data. Using the service could simplify the work of writing data management code.

The ORPG data query software implements only a small subset of the functionality of a typical database system. It does not support advanced features such as complex joint query and batch jobs, advanced report generation, GUI development support and others. If one needs those features, one should consider a commercial or GNU data base.

The ORPG data query provides the key functionality of querying a large collection of entries. It offers the applications the convenience and reliability that commercial data base systems provide. It is, however, much lighter than a full-blown data base system. It does not have the licensing and upgrade cost associate with a commercial data base system. It does not require the level of maintenance, training and administration of full featured data base systems.

The ORPG data query is much more efficient than a typical data base system because query records are all stored in memory and efficient index trees are built for the records. It eliminates overhead processing needed by full featured data base system. For example, the ORPG data query can manage real time radar radial data without consuming substantial system resource.

4. SYSTEM MANAGEMENT

The ORPG is a loosely-coupled multi-tasking system. It consists of nearly one hundred processes and as many data stores which are being updated dynamically for publishing and exchanging data. We discuss here system management issues such as system start-up and shutdown, process monitoring and system status tracking.

In the ORPG, each task is designed, developed and tested separately. The interaction with other tasks are minimized and well-defined. The interaction is performed in a loosely coupled way: Each task reads its inputs from and writes its outputs to well-known data stores without making connections (such as pipes or sockets) to other applications. The loosely coupled architecture promotes isolation between tasks and thus offers several advantages. As long as a task's inputs and outputs are designed, it can be developed independently of other tasks. It can be tested with a test data feed even if the task that is supposed to provide the input is not available. When a new task needs to be added later, existing tasks do not need to be changed to provide data to the new task through new connections. Well-isolated tasks can be more reusable because it has less dependencies on other tasks. Loosely coupled tasks can be restarted without disturbing tasks that interact with it. This increases system robustness. Locating software defects becomes easier because each system function is often involved in only one task and its input and output can be easily tracked. In a tightly coupled system, such as one using remote procedure calls, a task will not run without other tasks that implement the remote functions. It would be more difficult to determine the problem area when errors occur. Problem analysis is also made difficult because data flowing through pipes, sockets and RPC protocol are hard to track and examine.

The nice features of a loosely coupled system come with a cost. Using files for inter-process communication is an old technology. Indirect communication can be inefficient and tedious to program. The solution to this is using LBs instead of plain files. The LB technology extends the file based communication idea in several ways and makes it possible to use it as a primary IPC mechanism in a complex system such as ORPG: It simplifies the programming API. It extends it to be applicable for high data rate flow (such as multiple streams of radar base data). It supports remote access in a cluster environment.

Another issue with the loosely coupled system is that we have to manage the processes and the data stores. Data stores have to be created and cleaned up. Processes have to be started, monitored and shut down. Manually starting one hundred processes can be a difficult job. Many systems use so called "glue" script, such as Perl or Python, to do this job. In ORPG, we use tasks written in C to perform the job. The decision is made because of the following: Most of the ORPG tasks are written in C. The benefit of adding another language to the system may not justify the cost. Management tasks written in C are more efficient and are able to take full advantage of the infrastructure software support. They have the same look and feel like other ORPG tasks. And, finally, those "glue" scripting tools do not provide strong support for task management and monitoring in the ORPG cluster (multiple node) environment.

The tasks for ORPG management are the following.

1. "Manage RPG" (mrpg). This is the main task that sets up the ORPG run-time environment (creating and initializing data stores, generating and distributing the system configuration file and so on), starts (shuts down) operational tasks, monitors operational processes and maintains ORPG states. There is only one instance of mrpg running. In the ORPG loosely coupled system, mrpg is just one of the operational tasks. It is not a master process. It can be terminated and restarted while ORPG is operating. No other task is affected if mrpg is stopped except the service provided by mrpg becomes unavailable.

2. "Node Data Service" (nds). This is a process that collects the status of the ORPG operational tasks. The status is collected in near real-time (thus must be implemented efficiently) and reported to mrpg. An instance of nds runs on each node. mrpg starts nds as needed.

3. "RPG Process Status" (rpg_ps). This is a tool for displaying process status of all ORPG operational processes.

4. "Operate RPG" (operate_rpg). This is a task that helps to start the RPG automatically after the operating system boots up. It also helps to implement certain operational security features.

mrpg performs the procedures of starting up and shutting down the ORPG system. The following steps are performed to start the RPG.

1. Reads RPG configuration information from various configuration files such as the task table, product table, data table, comms configuration and resource allocation table.

2. Generates the ORPG system configuration file and distributes it to all nodes. This file contains the physical locations of all public data stores.

3. Creates all public product and data stores. If a data store exists and does not need to be recreated, its attributes are verified.

4. Executes a list of commands defined in a configuration file to initialize the data stores. Many adaptation and status data stores need to be initialized. The initialization commands are invoked in defined sequence and one after another. If a critical command fails, the startup procedure fails and the error is reported.

5. Starts all ORPG operational tasks (processes). All operational processes run concurrently as background processes. There is no defined start up order. mrpg can choose to start them in any order. If a process cannot be started because, for example, the executable file does not exist, the startup procedure fails and the error is reported.

mrpg creates data stores and starts processes on the correct node according to the resource allocation configuration.

mrpg accepts and processes other commands such as shutting down ORPG, putting ORPG in stand-by mode, turning ORPG into test mode and so on. One can issue a command by directly running mrpg. e.g. “mrpg startup”. One can also send a command to the mrpg command queue. All commands in the queue are processed sequentially.

mrpg monitors all operational processes. Any failed process is reported immediately. mrpg can restart any failed or killed process if it is specified to do so. Any duplicated process is detected and killed. Memory and CPU utilization of processes are monitored. If a process is detected to consume excessive memory or CPU resource, it is killed.

Operational processes running in ORPG do not need to report to mrpg. Nor do they need to be implemented to call specific ORPG functions in order to be incorporated into ORPG. That is, a third party application not developed for the ORPG can be used as an operational process. Such a process, of course, will not take advantage of the ORPG infrastructure support. Operational processes do not need to be started from mrpg. The developer or operator can start, or terminate, any of them manually through command line. Any ORPG application, e.g. hci, can start a process too.

5. ACKNOWLEDGMENTS

ORPG developers, Hoyt Burchan, Eddie Forren, Gary Gookin, Aamir Nawaz, David Priegnitz, Dan Suppes, and John Thompson made numerous contributions to the ORPG software design.

6. REFERENCES

Jain, M., Jing, Z., Zahrai, A., Dodson A., Burchan H., Priegnitz D. and Smith S., 1997: Software Architecture of the Nexrad Open Systems Radar Product Generator (RPG). Preprints for 13th International Conference on IIPS, Long Beach, Ca, February 1997, pp 238-241.

Jing, Z., Smith S., Jain, M. and Zahrai, A.: Migration of Legacy WSR-88D Algorithms and Product Generators to the Open Systems RPG. Preprints for 13th International Conference on IIPS, Long Beach, Ca, February 1997, pp 245-248.

Jing, Z., Jain, M., and Burchan, H.: "User Profile Based Product Distribution in the WSR-88D Open Systems Radar Product Generator (ORPG). Preprints for 14th International Conference on IIPS, Phoenix, Arizona, January 11-16, 1998. pp 238-241.

Jing, Z. and Jain, M.: The Linear Buffer and Its Role in the WSR-88D Open System RPG. Preprints for 15th International Conference on IIPS, Long Beach, Ca, January 11-16, 2000, pp 395-398.