17

COMMAND AND CONTROL

PROBLEM DESCRIPTION

Command and control is the classic feedback control loop that is seen in most engineering applications. A *feedback control loop* is a continuous process of state measurements, analysis, decisions, and commands in an effort to change the state of the system. Typically the goal of the analysis and commands is to keep the system in a stable, well-known state commonly referred to as a "steady state."

I will discuss two basic applications that share one common characteristic; the systems consist of many devices (e.g., computers, sensors) with the requirement to collect information from the dispersed devices and perform an analysis on the collected information. Command-and-control loops continue the process of immediately processing and analyzing the data, with the resulting adjustment of the system state as the final outcome. Figure 17.1 illustrates this state management process for the command-and-control loop function.

We can consider a subset of command-and-control loops to exist when the control commands of the feedback loop are not needed. Monitoring weather, for example, utilizes a vast network of sensors, weather stations, and other facilities. The information from the sensors is recording weather conditions for analysis. The speed with which the data are collected and analyzed has a direct impact on the required actions that need to be taken depending on the final results. However, in some cases the resulting actions may not necessarily be to issue a command back to the sensors but rather to issue a storm warning to the public.

Distributed Data Management for Grid Computing, by Michael Di Stefano

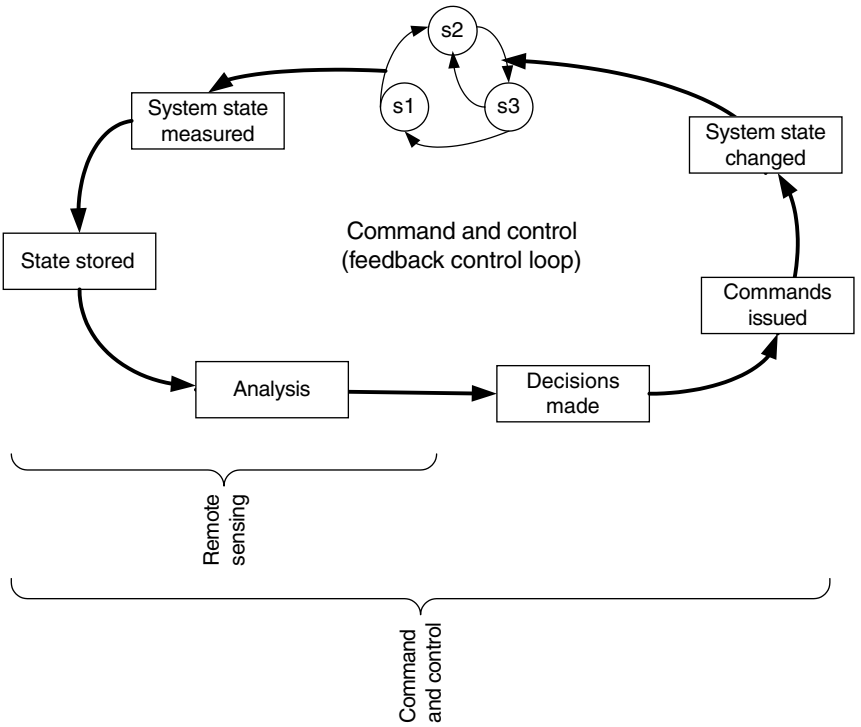

Figure 17.1. Command-and-control loop flow.

In the following discussions, the terms *sensor*, *device*, and *computer* are used interchangeably.

SOLUTION ARCHITECTURE

There are two possible architecture implementations for the command-and-control (command/control) loops that will be discussed and compared. In the first case, I will analyze the process where there is no data grid and then expand to the use of the data grid in the second case. The procedural steps in command/control loops are independent of the implementation, therefore enabling a direct comparison to the efficiency gained by using a data grid. The procedural steps normally outlined for the command/control loops are as follows:

1. Collection of data from the remote devices
2. Preparing the data for analysis, formatting, and inserting the data into storage
3. Analysis of the data
4. Decision process with input from the analysis process
5. Formulation of commands supporting the decisions of step 4

6. Issuing of commands
7. Delivery of commands out to the respective remote sensors
8. Repetition of the process, thus looping back to step 1

Command and Control Without a Data Grid

In step 1 as identified above, the collection of data from the sensors is always a good place to start. In a homogeneous sensor environment, the sensors are supplied by the same manufacturer or have a standard interface on which all manufacturers agreed. Therefore the data collection processes are uniform. However, in the more realistic scenario the sensor environment is heterogeneous and for each sensor type, manufacturer, and interface there are multiple sensor interfaces and data collection processes. The sensor collects data that need to be shared with other parts of the system for processing. To extract the data, one must know how to physically connect to the sensor, understand its communication protocol, and understand its data format. With this information and the knowledge of the larger systems networking characteristics, "adapters" must be created to collect the data from the sensor and deliver them to a common storage for analysis. Conversely, this same process must be done in reverse in order to send commands (if necessary) back to the sensors. Most likely, the input data process to the sensor will be different from its output process. For the purposes of simplifying this discussion, the physical connectivity for input data is the same for output sensor data. However, there could be a separate protocol and data format for the input process and data stream. Keep in mind that

- Data are moved from one system to another; the message that is transported over the network must be marshaled before transmission and then unmarshaled on receipt.
- In a heterogeneous environment, the input/output data process must be duplicated for each different sensor type, manufacturer, and interface.

Once the sensor connectivity is established, the collected data must be stored somewhere on their receipt. For example, the storage facility can be a file system, database, or even the memory space of the analysis process itself. Odds are that the data format from the sensor will not be the same as the input data format required by the analysis process. Therefore a level of data translation must take place. If the data are to first be placed in a file system or database for later retrieval by the analysis process, then it is safe to assume that the data representation of the storage medium is different from that of the analysis process. Therefore the sensor data must go through two data translation processes: sensor to file and file to analysis. As the process continues, one can see the large number of required data transformations that take place. Figure 17.2 shows this process without the use of the data grid.

In Figure 17.2, the triangles represent the sensors that interface to the data collection process and command distribution process. Once the data are collected and stored, the analysis process begins, followed by the decision and finally the

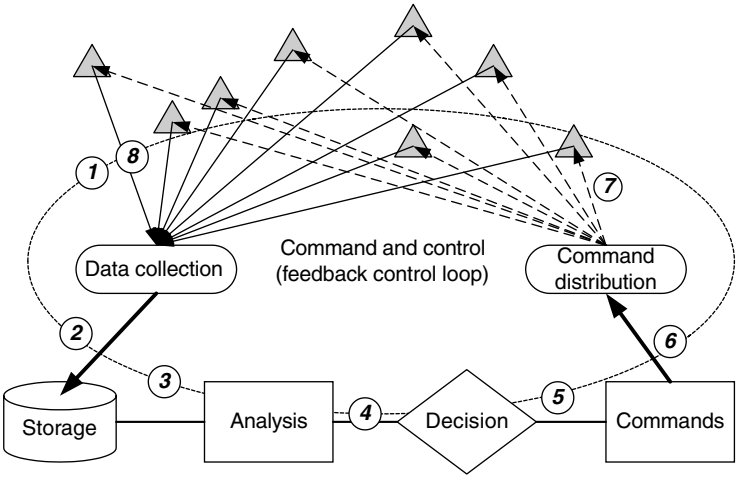

Figure 17.2. Command-and-control loop without a data grid.

formulation of the command. The command is than sent out to the external sensor environment. Physically, the command/control loop system may be a single system that we are representing as three logical components. Analysis is performed directly against stored data. The analysis process extracts the input data from the data store and performs its operations with the resulting output feeding the decision process. Commands are then issued through a communication mechanism specific to the topology of the underlying communications network of the command center and the sensors. The process of issuing the commands to each individual sensor is similar to the one described for the data collection, which involves connectivity, protocol, marshaling and unmarshaling, and data format translations.

Command and Control with a Data Grid

Applying the same system as described above, the data grid architecture simplifies the process, eliminating the number of "moving parts." The data grid encompasses the sensors and the command center where the analysis/decision/command process occurs. The sensors place or put their data directly into the data grid, therefore making them immediately available to the analysis process, completely bypassing the storage step and many of the data translation steps required without the data grid as described in the other scenario. In addition, the data grid can also serve as the storage medium, unless the raw sensor data need to be persisted in a database for long-term or future reanalysis. Should long-term persistence be required, the data grid can assume the responsibility of persisting the data to the proper storage device (e.g., file, database), a process independent of the analysis and command paths.

The result is the elimination of the multistep process of collection, local store, packaging, download, unpackaging, storing, and finally reading into the analysis programs with a single step of "writing to a data grid." The issuing of commands

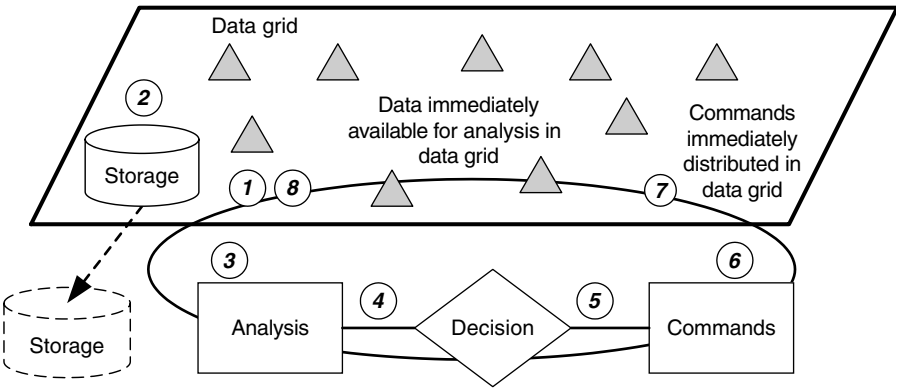

Figure 17.3. Command and control with a data grid.

can also be done via the data grid. Commands can be put into an area of the data grid from which the remote sensors can read. Again, taking the diagram of the command/control loops as represented in Figure 17.2 and architecting with the data grid results in the scheme shown in Figure 17.3.

Observations and Comparisons

The key points of differentiation between the data grid and non-data-grid implementations are data collection, translation, availability, and finally connectivity for transport of information to and from the sensors. Now the workflow process is simplified to the schematic representation in Figure 17.4, with the data grid as an integral part of the system.

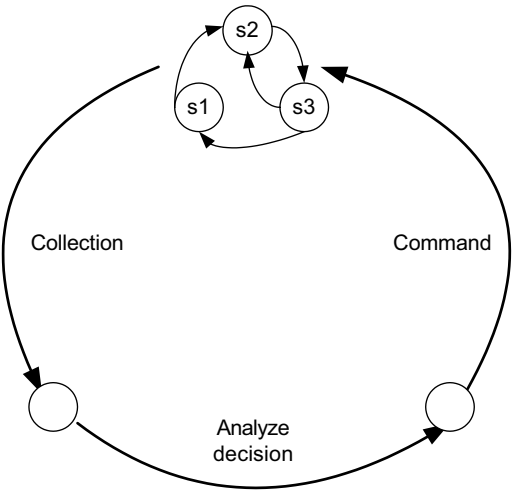

Figure 17.4. Points of comparison.

TABLE 17.1. Points of Comparison Between Data Grid and Non-Data-Grid Implementations

Point of comparison	Procedure steps	Non-data grid implementation	Data grid implementation
Collection	Collect data	Yes	Yes
	Translate data	Yes	Yes
	Transport data to central store	Yes	N/A
	Translate data for database	Yes	N/A
	Store in database	Yes	N/A
Analysis	Analyze	Yes	Yes
	Decision	Yes	Yes
Command	Store	Yes	Yes
	Forward	Yes	N/A

Without a data grid, the solution is highly customized and an artistic process. The variable parameters of the process are sensor type, location, data collection and storage at the sensor, and connectivity to the rest of the network. For example, if the sensor is a Unix computer and you are interested in its operational state, then the information in the "syslog file" is of value. Data collection for these data can be via a customized protocol or a well-known one such as a FTP. This is a non-real-time batch process where data are accumulated and periodically downloaded to a central repository.

Issuing of commands back to the sensors is a separate process but can leverage similar methods used in data collection. For example, the command files are FTPed so as to be read by the sensor; or command messages may be passed via a networked messaging or queuing protocol. The point is that these are all customized methods of collecting and issuing commands between sensors and the control center and are usually nonstandardized.

A side-by-side comparison of data grid versus non-data-grid implementations of command/control loops shows that there is an elimination of the complex moving parts once the data grid is architected in the solution. Thus, there is an increased efficiency since the time is drastically reduced from the point that data are available from the sensors for analysis. Table 17.1 identifies the various components that are needed in the two scenarios:

DATA GRID ANALYSIS

I will make some assumptions regarding the topology of the command/control system that we will analyze as part of this exercise. The complete sensor community is large in number and distributed across large geographic areas. At each location there is only one or a small number of collection sensors. The sensors are heterogeneous in nature as outlined in the earlier discussion in this chapter. The data atom size for sensor input and output is small in size, let us say on the order of

100 bits, and the data transfer intervals are on the order of minutes. However, this is just one part of the system processing requirements. The second part revolves around the data analysis and the command process. Even though each sensor produces a small amount of data per update, the sensors are large in number and the total sum of the data that needs to be analyzed, in sum, is quite large. The analysis process can be quite complex, thus making it very similar to the data mining and data warehouse use case as highlighted in earlier chapters. In this situation, the data collection aspects of the system are very similar to those of the geographic boundary use case of Chapter 16. Both are presented below with some customization for the particular case at hand, command/control loops.

The previous example for the data warehouse and OLAP analysis is as follows:

- *Application Definition Equation for the Distributed Environment*

$$OLAPProcess\begin{pmatrix} Work(), \\ Data\_output(), Data\_S1(), \ldots Data\_Sn(), \\ Time(), \\ Geography(), \\ Query() \end{pmatrix}$$

where

$$Work(atomic, nonsynchronous)$$

The output data surfaces are smaller for the command/control loops in comparison to the data surfaces that will be analyzed from the data warehouse.

$$Data\_output(\text{“}x\text{”}\ kbits, \text{“}y\text{”}\ bits, transactional, nontransient, queryable)$$
$$Data\_S1(\text{“}z\text{”}\ Tbits, \text{“}k\text{”}\ Mbits, nontransactional, transient, queryable)$$
$$Data\_Sn(\text{“}z\text{”}\ Tbits, \text{“}k\text{”}\ Mbits, nontransactional, transient, queryable)$$

The ability to run complex analysis over large data sets in short periods of time provides the command/control process with a better view into the system state, demands on the system, including its current state, and its ability to meet the demands. Armed with better and in-depth quality views, the command/control process can target specific “in-time” adjustments to the system to meet the demands placed on it. This command/control OLAP process is to run in the confines of a single data center, and the applications requirement to analyze (complex queries) any of the data sets is not essential to the business.

$$Time(Near\text{-}Real\text{-}Time)$$
$$Geography(DataCenter, 1GbitEthernet)$$
$$Query(complex)$$

- *Data Management Policies*

$$DataDistributionPolicy = DDP\left(\begin{array}{l} DataWarehouse\_DDP, \\ DWRegion, \\ Scope(ALL), \\ Pattern\left(\begin{array}{l} Automatic, Random\left(\begin{array}{l} DWDDPPattern, \\ WhiteNoise(\,), \\ NULL, \\ NULL, \\ NULL, \\ NULL \end{array}\right) \end{array}\right) \end{array}\right)$$

The data replication policy shows a lower number of replicas per data atom as the overall size of the data in the data grid is quite large. Each replica increases the overall storage capacity of the data grid by the size of the data loaded from the data warehouse. The downside to a lower replication size per data atom is resilience of the data in case of a failure. These tradeoffs must be considered in each use case and architecture variation.

$$DataReplicationPolicy = DRP\left(\begin{array}{l} DataWarehouse\_DRP, \\ DWRegion, \\ 3, \\ Scope(ALL) \end{array}\right)$$

$$SynchronizationPolicy = SP\left(\begin{array}{l} DataWarehouse\_SP, \\ DWRegion, \\ Scope(Boundary(\text{“}intra\text{”}), NULL), \\ Transactionality(\text{“}nontransactional\text{”}), \\ LoadStore(List(\text{“}DataWarehouse\_DLP\text{”}), NULL), \\ Events(NULL) \end{array}\right)$$

Use of events to coordinate data consistency between the command/control loops and the data grid are managed via events or triggers from within the data grid to or from the sensors. This assumes that the EII functions of the system remain as part of the command/control loops. Should one choose to transfer the EII responsibility from the command/control loops to the data grid, then the data grid event notification policies will need to be established to maintain data consistency.

$$EventNotificationPolicy = N/A$$

The data load policy is required in this use case, and if the implementation offers the business application, the data grid as the medium to data loads and data pushes to the sensors. Then the data load policy will manage the sensors'

interface of extract and load result sets into the data grid.

$$DataLoadPolicy = DLP\begin{pmatrix} DataWarehouse\_DLP, \\ MCRegion, \\ Granularity(Grouping(1), N/A), \\ DataWarehouseAdapter( ) \end{pmatrix}$$

$$DataStorePolicy = N/A$$

Geographic boundary analysis may be applied to the command-and-control loops:

- *Application Definition Equation for a Distributed Environment*

$$GeoBoundaryProcess\begin{pmatrix} Work(atomic, nonsynchronus), \\ Data(MultiGbits,100bits, nontransactional, \\ \quad transient, queryable), \\ Time(near\text{-}Real\text{-}Time), \\ Geography(WAN), \\ Query(Basic) \end{pmatrix}$$

- *Data Management Policies.* The data distribution and replication policies for this scenario may need to be a manual pattern as opposed to an automatic one. While each sensor is a node in the data grid, it is safe to assume that it will not be available to contribute storage capacity to the data grid, thus eliminating it as a possible data replication and distribution node for other sensor data. However, for this analysis, we will use the same white noise distribution policy as before. It is left to the reader as an exercise to determine the physical and behavioral characteristics of the overall system and to create a distribution pattern to meet the specific requirements:

$$DataDistributionPolicy = DDP\begin{pmatrix} GeoBoundary\_DDP, \\ GBRegion, \\ Scope(ALL), \\ Pattern\begin{pmatrix} Automatic, Random\begin{pmatrix} DWDDPPattern, \\ WhiteNoise( ), \\ NULL, \\ NULL, \\ NULL, \\ NULL \end{pmatrix} \end{pmatrix} \end{pmatrix}$$

The data replication policy for the commandcontrol loops as illustrated above shows a low number of replicas per data atom in order to meet the assumptions

made in the data distribution policy expression:

$$DataReplicationPolicy = DRP\begin{pmatrix} GeoBoundary\_DRP, \\ GBRegion, \\ 3, \\ Scope(ALL) \end{pmatrix}$$

$$SynchronizationPolicy = SP\begin{pmatrix} GeoBoundary\_SP, \\ GBRegion, \\ Scope(Boundary(\text{“}inter\text{”}), NULL), \\ Transactionality(\text{“}nontransactional\text{”}), \\ LoadStore\begin{pmatrix} List(\text{“}DLP\_SensorTypeA\text{”}, \ldots), \\ List(\text{“}DSP\_SensorTypeA\text{”}, \ldots) \end{pmatrix}, \\ Events(List(\text{“}ENP\_SensorTypeA\text{”}, \ldots)) \end{pmatrix}$$

Event notification may not be necessary in this application for data gathering; however, it will prove valuable to notify nodes that there are commands waiting for them to be read in and acted on:

$$EventNotificationPolicy = ENP\begin{pmatrix} ENP\_SensorTypeA, \\ GBRegion, \\ Scope(\text{“}All\text{”}), \\ SensorCommandFunction() \end{pmatrix}$$

Similar load policies are needed for each sensor type. Only one example is illustrated below for all the possible types of sensors:

$$DataLoadPolicy = DLP\begin{pmatrix} DLP\_SensorTypeA, \\ GBRegion, \\ Granularity(Grouping(1),10000), \\ Adapter(\text{“}SensorA\text{-}OutputAdapter\text{”}) \end{pmatrix}$$

Similar store policies are needed for each sensor type. Only one is illustrated below for the possible types of sensors:

$$DataStorePolicy = DSP\begin{pmatrix} DSP\_SensorTypeA, \\ GBRegion, \\ Granularity(Grouping(1),10000), \\ Operation(\text{“}store\text{”}), \\ Adapter(\text{“}SensorA\text{-}InputAdapter\text{”}) \end{pmatrix}$$

- *Qos–Application Requirement Quadrant Graph*. As can be seen from Figure 17.5, the command-and-control loop falls into a level 1 zone and is atomic in nature.

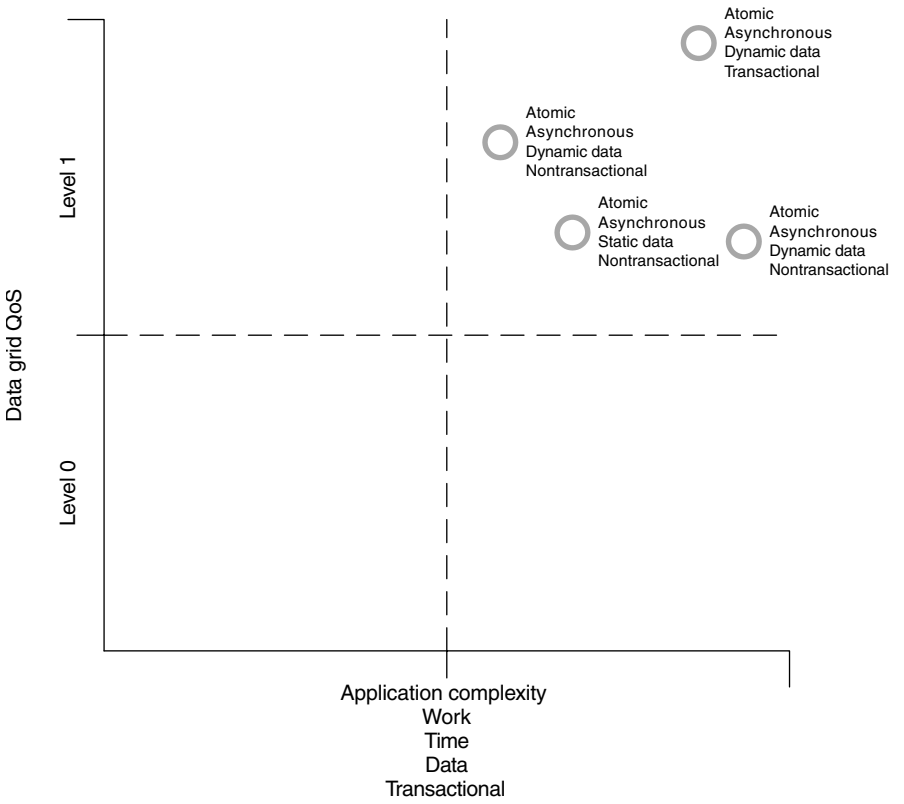

Figure 17.5. Command and control.

APPLICATION SPINOFFS

Command and control plays a key role in a compute utility service. In order for a computer utility service to be effective, the state of the service must meet the supply-and-demand curves of the user community. As the need for the computer services increases, the utility has to be able to adjust to the demand requests by changing its profile; thus, reallocation of the physical resources is necessary. This implies that information describing the state of the utility service including user demands is monitored on a real-time or near-real-time basis. Analysis must be done on this information and commands must be issued back to the utility service for it to change its state in a timely fashion in order to meet the demand on the system at that point in time.

Should the change in state of the utility service lag behind the demand, the utility service will be put into a state that does not meet the demand of the user community. If this happens repeatedly, the utility service will move out of a steady-state condition to one that adversely affects its quality of service to the customer.

A second example is the collection of information from remote devices, where the analysis needs to be performed in real time. However, no commands are issued back to the system to change its state; instead the results that have been analyzed are used elsewhere, for example, by another external system. Weather monitoring entails the ability to collect data from remote weather stations, analyze the raw data in near real time, and have the results available to people or other computer systems to generate weather alerts such as tornado and hurricane warnings.