

# Autonomic WWW Server Management with Distributed Resources

Takuya Araki

NEC Internet Systems Research Laboratories / Argonne National Laboratory

t-araki@dc.jp.nec.com

## ABSTRACT

If many people access a Web server at one time, the server might not be able to respond within an acceptable time or even provide the service. Therefore, enough servers should be assigned to a service to guarantee quality of service. But reserving a lot of resources for peak access is not cost effective, because these resources are idle most of the time.

In order to solve this problem, technologies called utility computing or autonomic computing have been proposed and are under development. However, these technologies utilize resources only within one organization.

In this paper, we present an autonomic system architecture that uses distributed resources leveraged by Grid technology. In our architecture, computing resources are rented from different organizations. Our architecture supports the J2EE system; hence, existing Web applications can be used without any modification. In addition, our architecture considers the location of the resources when redirecting a request to a server and allocating a new server, thereby leading to better performance. We adopted WS-Agreement as an interface for negotiating service level agreements.

We have implemented and evaluated this system and confirmed the effectiveness of this architecture.

## 1. INTRODUCTION

The reliability of Web servers is important. If many people access a server at one time, the server might not be able to respond to the client within an acceptable time or even provide the service. Therefore, enough servers should be assigned to a service to guarantee quality of service. But reserving a lot of resources for peak access is not cost effective, because these resources are idle most of the time.

In order to solve this problem, technologies called utility computing or autonomic computing have been proposed and are under development [8, 10]. With these technologies,

servers are automatically added or removed from a service according to the load of the service. Since the resources can be shared between services, these technologies can make the utility rate of the servers higher.

However, these technologies are not enough to reduce the cost of servers because they share resources only within one organization. Thus, many resources might be left idle if there is not enough work.

In this paper, we present an autonomic system architecture that uses distributed resources leveraged by Grid technology; computing resources are rented from different organizations. In addition, the accesses from clients are redirected to a server based on the distance between them. This strategy improves the performance of Web accesses.

The contribution of this paper is a novel architecture of distributed autonomic computing that includes following characteristics:

**Support** Our architecture supports J2EE system, which means that not only simple static Web pages but also complex Web systems such as a shopping site can be supported by our system.

**Security** Security is the most important issue when utilizing distributed resources; this issue does not arise in the current utility or autonomic computing technology. Our architecture uses GSI [7] for authentication and dynamically creates VPN for secure communication.

**Interface** We use a standard interface for our architecture. Specifically, the latest draft of WS-Agreement [2] is used for an interface for negotiating service level agreements and resource reservations. We believe this is one of the early attempts to use the WS-Agreement specification.

**Location Awareness** Accesses from clients are redirected to one of the distributed servers based on the location. In addition, when a new server is added, the location of the server is based on the history of access. We believe location awareness is the key feature of distributed autonomic computing system.

We envision that our system will produce new business opportunities: a resource provider as a resource seller, and an

autonomic Web server manager as a kind of a value-added resource reseller.

The rest of this paper is structured as follows. Section 2 explains the background and requirements of the system. Section 3 shows the architecture and its implementation. Section 4 presents a preliminary evaluation on the system. Section 5 discusses related work. Section 6 briefly outlines future work.

## 2. BACKGROUND AND REQUIREMENTS

In this section, we describe the technology of the Web system as the background and requirements to our system.

### 2.1 Background

As we mentioned, we target not only simple static WWW sites, but also dynamic sites like shopping sites. Such dynamic sites are usually composed of multiple tiers.

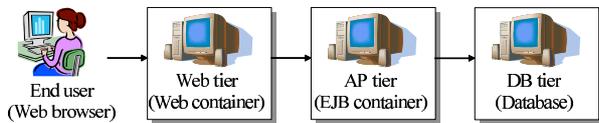


Figure 1: Three tiered architecture

Figure 1 shows a three tiered architecture of a Web system, comprising a Web tier, an AP tier, and a database (DB) tier. The Web tier is used mainly for creating HTML documents. The AP tier is responsible for business logic, which is independent of the presentation layer (the Web tier). The DB tier stores information that is used by the AP tier, such as stock information for a shopping site. It is possible to merge or subdivide some of the tiers, but basically a Web system can be understood by this model.

J2EE (Java 2 Platform, Enterprise Edition) supports this kind of Web system. In J2EE, the Web tier is implemented as a Web container running servlet, and the AP tier is implemented as an EJB container running EJBs (Enterprise Java Beans). Please refer to [3] for further information about J2EE.

### 2.2 Requirements

Several requirements guided our design of an autonomic system architecture. Most important, our system should be able to support the three tier model, especially the J2EE system. Because a lot of Web systems are already written using J2EE, it is desirable that existing Web applications can be used without modification.

The system should make it possible to add or remove distributed resources in order to keep quality of service. And, as we mentioned before, location of resources should be taken into account when redirecting user's request and adding a new resource.

Furthermore, communication between distributed resources should be secure, since it may contain secret information like credit card number.

## 3. ARCHITECTURE AND IMPLEMENTATION

In this section, we describe the architecture and implementation of the system designed to meet the requirements described in the previous section.

### 3.1 Overall Structure

Figure 2 shows the overall structure of the system.

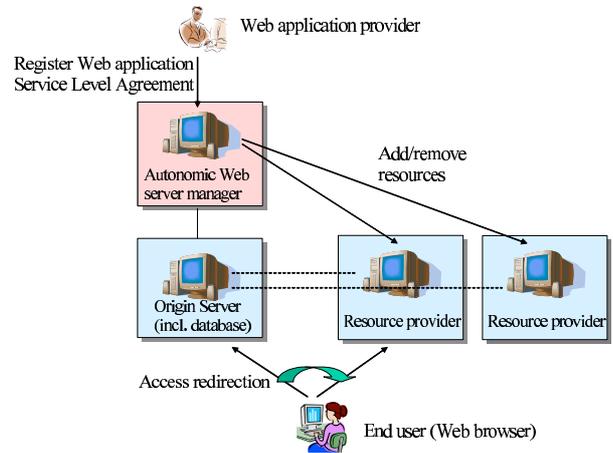


Figure 2: Architecture of the system

The Web application provider provides Web applications like “ear file” (if the system is using J2EE). It also negotiates service level agreements with the autonomic Web server manager. This interface uses WS-Agreement.

The autonomic Web server manager manages the addition and removal of resources and redirects access from end users.

The resource provider provides computational resources. This interface is also using WS-Agreement, and the resource can be reserved for certain period.

The origin server is one of the computational resources provided by a resource provider, but it contains database; it is not removed even when the load is low. Other computational resources provide a Web tier and/or AP tier.

The access from the end user is redirected to one of the resources, considering the distance to the resource, load of it, and so on.

We note that all the entities can belong to different organizations: the Web application provider, the autonomic Web server manager, resource providers, and end users. Resource providers lease their resources to other organizations. The autonomic Web server manager rents these resources to provide an abstract Web server that guarantees quality of service. The Web application provider rents the abstract Web server from the autonomic Web server manager.

### 3.2 Detailed Structure

Figure 3 shows the detailed structure of the system. We explain below each of the services and modules.

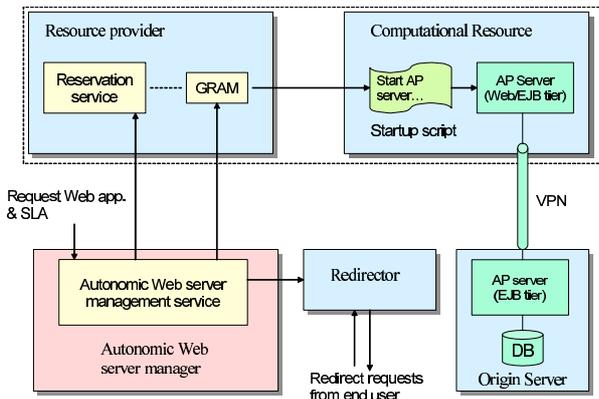


Figure 3: Detailed structure

### 3.2.1 Resource Provider

Our architecture uses Globus Toolkit 3.2 (GT3.2) [15]; therefore, all services are written as Grid services. GSI [7] is used for authentication.

In order to reserve a computational resource, the reservation service is called. It manages the table of reservation states. After the resource is reserved, GRAM can be used to submit a job. In order to enforce reservation, the system should ensure that GRAM is used by an entity that is not reserving the resource. Not to do this, the reservation service modifies grid-mapfile; if the resource is reserved by an entity, the reservation service changes the grid-mapfile to include only the entity. After the reservation period, the grid-mapfile reverts to the original form. After the reservation, currently running jobs are canceled by the reservation service.

If GRAM is set up to use a batch job system that supports advance reservation, the reservation service can utilize that. For example, MAUI [5] has a command `setres` that allows users to reserve the system. In this case, the reservation service need not modify the grid-mapfile. Submitted jobs just remain in a queue if the resource is reserved by another entity.

We decided to use GRAM directly instead of creating a special service such as Web server service. Our reason was that we wanted to make the burden of resource providers smaller; in our architecture, resource providers need to install only GT3.2 and the reservation service. In addition, the computational resource can execute any kind of job other than the AP server. It is important to increase the utility rate of the computational resource.

For the interface, the reservation service uses the document format specified in the current draft WS-Agreement. However, it was not possible to comply with the protocol described in the specification, because it depends on WS-RF, which is not supported by GT3.2.

The document format of the WS-Agreement specification specifies only the container of domainspecific agreement information. Therefore, the domainspecific part of the agreement document must be specified. To this end, we extended `ServiceDescriptionTermType` to include reservation infor-

mation like reservation period and number of nodes to reserve.

Renegotiation is not supported by the current implementation. We will implement it in the future because it is important for usability.

### 3.2.2 Startup Script

An AP server is invoked from a startup script executed by GRAM. The Web application program is transferred in the startup script using GSI-SCP and is then deployed to the AP server. (GSI-SSH and GSI-SCP are extensions of SSH and SCP that can use GSI for authentication.)

In addition, a VPN connection to the origin server is created in the startup script. This ensures that the communication between the origin server and the allocated resource is secure and transparent. To create VPN, one can choose from several methods including VTun, OpenVPN, and PPP over SSH. We used PPP over GSI-SSH in order to use GSI in the current implementation.

Admittedly, security can be preserved without using VPN. For example, GT3.2 has a functionality to wrap an EJB as a Grid service whose access can be made secure. However, it requires modification of the Web application, which is not desirable considering the requirements discussed in the previous section.

In the evaluation system, we used Jboss 3.2.3 [11] for the AP server. Jboss is a free implementation of J2EE specification. We used PostgreSQL 7.4.2 for the database.

For the Web application, we used Java Petstore 1.1.2 [14], which is an example J2EE application developed by Sun Microsystems.

### 3.2.3 Origin Server

As we noted, the origin server contains a database in our architecture. But how much functionality the origin server covers other than the database depends on the case.

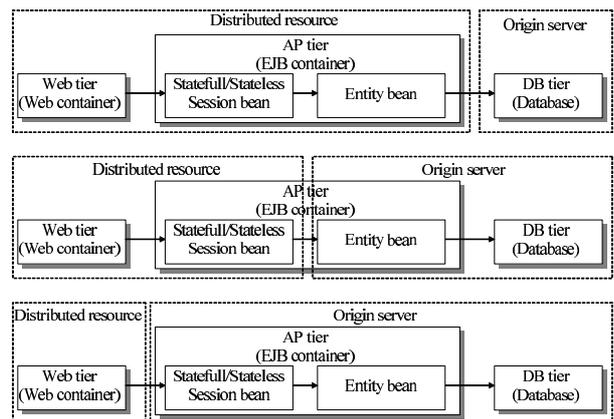


Figure 4: Functionality of the origin server

Figure 4 shows three cases. In this figure, we assume a J2EE system. In J2EE, the AP tier is implemented as an

EJB container, and there are three kinds of EJB: stateless session bean, stateful session bean, and entity bean. Stateful/stateless session beans are used to implement business logic, and entity beans are used to abstract database access; basically, one entity bean corresponds to a row of a database table.

In the top of the figure, the origin server runs only the database. In the middle of the figure, the origin server runs entity beans and the database. In the bottom of the figure, the origin server runs all entity beans and the database.

Each case has its pros and cons. In the first case, more work is done by distributed resources in parallel, which leads to better performance. However, directly communicating with database through the Internet may cause a performance bottleneck because the communication may contain transactions; transactions through a highlatency network may cause performance reduction because the transaction may lock a table of the database, and the duration of lock is prolonged by the network latency.

On the other hand, less work is done by distributed resources in the last case. However, transaction is managed by the AP server.

Therefore, which case should be used depends on the Web application especially the balance of computation and communication of the application.

In our architecture, we allow the application to decide which case to use. In the evaluation system, we distributed only the Web tier shown in the bottom of Figure 4.

### 3.2.4 *Autonomic Web Server Manager*

The autonomic Web server manager accepts a Web application and its service level agreement as a request.

The interface is similar to the reservation service; it uses the document format of WS-Agreement. We extended `ServiceDescriptionTermType` to include a Web application, and we extended `GuaranteeTermType` to include the required response time of the server and the URL path.

The autonomic Web server manager periodically measures the response time of the specified URL of each distributed resource. If the response time exceeds a certain percentage of the required response time (e.g., 80%), the autonomic Web server manager adds a new resource. If all response times are below a certain value (e.g., 500 ms), the autonomic Web server manager removes a resource. The policy used for resource addition and removal is described in Section 3.3.

If distributed resources are added or removed, the autonomic Web server manager changes the configuration of the redirector.

When a server is removed, it should be guaranteed that no end user is using the resource. Therefore, the system first stops redirection, and then, after a specified period, the server is actually removed. The Web application should ensure that after the specified period the session is invalidated.

### 3.2.5 *Redirector*

The purpose of the redirector is to redirect the access from an end user to an appropriate distributed resource. This technology is studied as distributed load balancing. One of the products related to this technology is found in [6].

This technology needs two different mechanisms: one to redirect the request transparently, and another to select the target of redirection.

To redirect the request transparently, DNS lookup or http redirection is used. When DNS lookup is used, the returning IP address of DNS lookup is set to that of the target host. When http redirection is used, the server returns “moved temporarily” status when an end user accesses a URL. The end user then connects to the target host, which is given in the response automatically. Using http redirection is easier than using DNS lookup, but it supports only http, and the redirection is visible from an end user.

To select a target host, metrics such as distance from the end user to the server and the load of the server are used. There are several methods to measure the distance. One method is simply to use a ping program. ICMP echo is used in the program, and the roundtrip time can be measured. However, this method takes at least the roundtrip time to measure the distance, and ICMP echo might be ignored because of a security reason. Another method is to use routing information stored in routers. The implementation is described in [9]. To implement this method requires access privileges to the router where each server is located.

In our current system, we use http redirection with ping because of the ease of implementation. We used a servlet to implement this method, and it is also executed by JBoss on the origin server.

To measure the latency from the resources to the end user, the redirector uses GSI-SSH to let the resources call ping to the end user. The SSH connections between the redirector and the resources are kept alive, instead of establishing connections every time. This approach reduces redirection time. The evaluated latency is cached; end users in a same network are treated to have the same latency (network size is configurable).

## 3.3 **Management Policy**

The management policy used when redirecting a user request and adding or removing resources is very important because it directly affects the resource usage. Here, we discuss what kind of management policy would be optimal for this architecture.

Figure 5 shows an example of connections between the end users, distributed resources, and the origin server.

The response time observed by the end user can be described

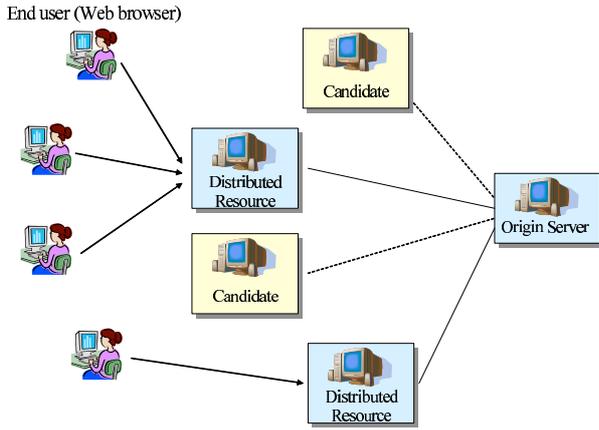


Figure 5: Example of connections

as follows:

$$\begin{aligned}
 ResponseTime = & Latency(EndUser, Resource) \\
 & + TransferTime(EndUser, Resource) \\
 & + Latency(Resource, Origin) \\
 & + TransferTime(Resource, Origin)
 \end{aligned}$$

Here,  $Latency(A, B)$  is network latency between A and B, and  $TransferTime(A, B)$  is contents transfer time from B to A.  $TransferTime$  depends not only on network bandwidth, but also on current speed of server B, which is a function of the original speed of B and current load of B.

The redirector should select a resource to minimize the *ResponseTime*.

When adding a new resource, a resource that reduces the average *ResponseTime* the most should be selected. To calculate the future average *ResponseTime*, one can use access history. This assumes that the future access tendency of the network locations of the end users will be the same as the previous access history. This applies when the contents are drawing interests from special locations (e.g., if the contents are written in Japanese, it is probable that most of the accesses come from Japan) or when a certain organization has a lot of end users (e.g., a large ISP).

If the response time from the user is below a certain threshold, one of the resources should be removed to reduce the number of rented resources. To select the resource to remove, one can apply the same method: a resource that increases the average *ResponseTime* the least should be selected.

Thus far we have discussed an ideal policy, and of course it is not possible to calculate above equation exactly. Approximation is needed. In our implementation, we assumed that  $TransferTime$  depends mainly on the current speed of the resource.

For redirection, the redirector uses the following equation:

$$\begin{aligned}
 ResponseTime = & a * Latency(EndUser, Resource) \\
 & + b * \frac{RedirectionCount}{RelativeSpeed} \\
 & + c * Latency(Resource, Origin)
 \end{aligned}$$

Here,  $RedirectionCount$  is number of redirected accesses to the resource within a certain period, and  $RelativeSpeed$  is relative speed of the resource;  $a$ ,  $b$ , and  $c$  are constants that are configurable.  $TransferTime(Resource, Origin)$  is removed from the equation, because it should be the same if it is approximated to be the function of the speed of the origin.  $RelativeSpeed/RedirectionCount$  represents the current speed of the resource, and  $b$  represents average amount of work; therefore,  $TransferTime$  is represented as  $b * RedirectionCount/RelativeSpeed$ .

For adding a resource, the above equation is also used. In this case,  $RedirectionCount$  is set to 1. To evaluate latency from the future end users to the resource, we use the history of the accesses as described before; all candidate resources are made to ping to recently accessed end users. The average is used as the *Latency*.

Currently the management policy is hard-coded in the program. Separating it to another module is a future task.

#### 4. PRELIMINARY EVALUATION

We conducted a preliminary evaluation of the system performance using one computer.

The server used for the evaluation had two Pentium Xeon 2.2 GHz processors and 2 GB memory, and it ran Red Hat Linux 7.3.

First, we evaluated the performance of redirection. This is an important metric because it directly affects the response time experienced by the end user.

Table 1: Redirection Time

w/ GSI-SSH Call	Reuse Connection	On Cache
427 ms	9 ms	1 ms

Table 1 shows the time of redirection. As we described before, the redirector lets distributed resources ping to the end user. In order to call ping, a GSI-SSH connection is established between the redirector and the resources. The first column of the table shows the redirection time including the time of establishing the connection. This only occurs at first redirection. The second column shows the redirection time when the connection is reused. The last column shows the redirection time when the ping value is on the cache. In this evaluation, we accessed from the local server and redirected to the same server (the above redirection time is evaluated at the server side). Since the time actually spent in calling ping is small (0.03 ms), it is negligible in this evaluation.

The evaluation shows that the cost of redirection is small enough. In particular, it shows that reusing the connection significantly reduces the cost of redirection. In an actual situation, the time of ping call is added to the redirection

time, but it can be controlled by using a timeout of ping command.

Next, we evaluated the cost of adding a new server.

**Table 2: Server addition time**

Total Time	Ping Time	JBoss Start Time	Rest
50.8 s	8.8 s	32.7 s	9.3 s

Table 2 shows the time of adding a server. The column of ping time shows the time which is used to call ping. It includes resource reservation time and GRAM job execution time for calling ping, and time of calling a callback program which tells the ping value to the system. The second column shows that more than half of the total time is spent for starting up JBoss. Rest of the time is used for miscellaneous things including calling a callback program which tells the system that JBoss has started up, and modifying the configuration of redirector. We believe that this is acceptable time for adding Web servers.

We also evaluated the cost of removing a server. The result, 20.1 seconds, is acceptable because removing a server is less time critical.

## 5. RELATED WORK

Several projects and products attempt to share resources between services. UDC (Utility Data Center) of HP [8], autonomic computing of IBM [10], Oracle 10g, Japanese Business Grid project [12] are examples. However, since their resource sharing is limited within a data center, they do not address the problems regarding distributed resources.

Polimatica [13] is a similar kind of system, but it emphasizes policy based management. It supports physically distributed resources but does not support resources owned by different organizations.

EdgeComputing [1] of Akamai has a similar structure to ours. It supports a J2EE system and utilizes distributed resources. However, it does not support resources administered by other organizations. Moreover, it does not support dynamic addition and removing resources.

JOSH [4] is a higherlevel job scheduler for Globus Toolkit 3. It has a functionality that uses ping to select a resource according to location.

## 6. CONCLUSION

We have presented a novel autonomic architecture for Web system. Leveraged by Grid technology, the architecture uses distributed resources. We described the detailed architecture, the current implementation, and the preliminary evaluation.

Future work includes evaluating the system in more detail and enhancing the system so that it can be used for a practical, largescale WWW system.

## 7. ACKNOWLEDGMENTS

This work was in part supported by the Mathematical, Information, and Computational Sciences Division subprogram

of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38. We thank Dr. Kate Keahey for useful discussion and suggestion.

## 8. REFERENCES

- [1] Akamai. A developer's guide to on-demand distributed computing. [http://www.akamai.com/en/resources/pdf/whitepapers/Akamai\\_DeveloperGuide\\_Distributed\\_Computing.pdf](http://www.akamai.com/en/resources/pdf/whitepapers/Akamai_DeveloperGuide_Distributed_Computing.pdf).
- [2] A. Andrieux, K. Czaajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web services agreement specification (WS-Agreement) version 1.1, 2004.
- [3] S. Bodoff, D. Green, K. Haase, E. Jendrock, M. Pawlan, and B. Stearns. *The J2EE Tutorial*. Addison-Wesley, 2002.
- [4] G. Cawood. Josh functional specification. <http://www.epcc.ed.ac.uk/sungrid/PUB/D4.1-FunctionalSpecification.pdf>.
- [5] Cluster Resources. Maui scheduler. <http://www.supercluster.org/maui/>.
- [6] F5 Networks. 3-DNS Controller. <http://www.f5.com/f5products/3dns/>.
- [7] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *5th ACM Conference on Computer and Communications Security Conference*, pages 83–92, 1998.
- [8] Hewlett-Packard. Utility data center: Overview. <http://www.hp.com/large/infrastructure/utilitydata/overview/>.
- [9] S. Horman. Globally distributed content (using BGP to take over the world), 2001. [http://www.supersparrow.org/ss\\_paper/stuff/ss\\_paper.pdf](http://www.supersparrow.org/ss_paper/stuff/ss_paper.pdf).
- [10] IBM. Autonomic computing. <http://www.research.ibm.com/autonomic/>.
- [11] Jboss Inc. Jboss: Professional open source. <http://www.jboss.org/>.
- [12] H. Kishimoto, T. Kojo, and F. Maciel. Business grid middleware: Goals and status, 2004. GlobusWORLD 2004, <http://www.globusworld.org/program/slides/3c.1.pdf>.
- [13] Y. Maeno, M. Kawato, S. Nisimura, F. Machida, H. Fujio, and T. Kamachi. Polimatica: Abstraction in policy-customizable private virtual organizations. In *IEEE International Conference on Web Services (IWCS'2004)*, 2004.
- [14] Sun Microsystems. Java Pet Store. <http://java.sun.com/developer/releases/petstore/>.
- [15] The Globus Alliance. The Globus Toolkit. <http://www.globus.org/toolkit/>.

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.