Enterprise Software as Service

Online services are changing the nature of software.

DEAN JACOBS, SALESFORCE.COM

While the practice of outsourcing business functions such as payroll has been around for decades, its realization as online software services has only recently become popular. In the online service model, a provider develops an application and operates the servers that host it. Customers access the application over the Internet using industry-standard browsers or Web services clients. A wide range of online applications, including e-mail, human resources, business analytics, CRM (customer relationship management), and ERP (enterprise resource planning), are available.

In the enterprise setting, online services compete with the in-house deployment of packaged software. The primary advantage of the online model, as with any form of outsourcing, is that it aggregates many users and can

Enterprise Software as Service

leverage economies of scale to reduce costs. This principle applies to all aspects of the IT infrastructure, including hardware, software, staffing, and the data center itself. A collateral advantage of online services is that the sharing of capacity enables pay-as-you-go pricing. This means up-front costs are low, customers pay only for what they use, and costs can be eliminated entirely by canceling the service.

An important leverage point for online services is that they vertically integrate development of the application with development of its supporting infrastructure, including file systems, databases, application servers, and management tools. Infrastructure components can therefore be tailored to meet the needs of both the particular application and the online model in general. For example, the Google file system is optimized to support reads and appends rather than arbitrary writes.[1] Specialization of the application management infrastructure can significantly reduce the need for IT staff, representing a significant chunk of operating costs.

Software as a service can be developed more easily than packaged software. In the online environment, an application is installed in only one setting; hence, there is no need to port it to a variety of platforms. Moreover, the service provider has direct control over when upgrades occur and, thus, which versions of the application need to be available at any given time. Finally, since the service provider manages the actual application instance that is used by customers, bugs are easier to reproduce and test cases can be developed against the actual data, within any privacy constraints.

An important issue in the design of an online service is the extent to which it can be customized. Customization is required for many complex business processes and can provide a business with a competitive advantage, but it can also mean that upgrading a service is harder. More importantly, customization can make it harder for an organization to set up and/or migrate away from a service. Whereas ease of setup and migration are not essential to the online business model, they reduce the risk of trying the service and offer a hedge against providers that fail.

The benefits of online services vary depending on the size of a customer's business. A large enterprise with its own data center can aggregate users and benefit from economies of scale. For small to medium-size businesses that lack the resources to set up and maintain a complex data center, online services can provide basic automation of business processes.

While the first online enterprise services appeared in the 1990s, the model has only recently begun to gain wide acceptance. One reason is that the Internet has matured: Web UI technology can produce a better browser experience; Web services protocols make it easier to integrate with external applications; service providers benefit from cheaper, higher-quality bandwidth; and IT managers are becoming more comfortable with the technology. Another factor is that service providers are now building online applications from the ground up so they have better security, better quality of service, and are more cost-effective. Service providers are also getting their businesses off the ground by targeting applications that are less mission-critical, such as campaign management and approval chain processing, at least initially.

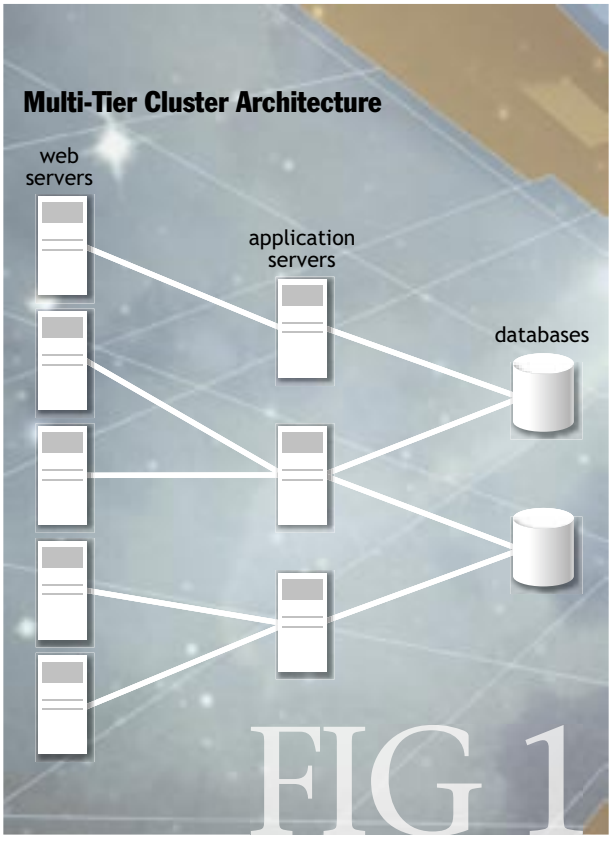
In this article, we look at software as a service and how it differs from traditional packaged software. First we compare these models in terms of their quality of service, including scalability, performance, availability, and reliability. Then we discuss the problem of leveraging economies of scale. We also look at application customization and online programming environments, as well as application integration and Web services. The article concludes with a strategy for evolving a successful online service.

QUALITY OF SERVICE

An online service must be designed from the beginning to be highly scalable. As an example, a moderate departmental application generally supports hundreds of users. A successful online implementation of this application

would have to handle thousands of such departments and, thus, hundreds of thousands of users. Loads of this magnitude require the use of clustering,[2] where groups of processes are spread out across multiple machines and coordinate their actions to provide the service. Scalability is offered by allowing the number of processes and machines to vary according to the presented load and by balancing requests across the system. Figure 1 illustrates a typical Web-based application where clustering is used throughout the system, from Web servers in the front end to application servers in the middle tier to databases in the back end.

To improve the reliability and availability of an online service, a service provider should employ clustering to eliminate single points of failure. A management infrastructure should be provided to monitor the health of server processes and either restart them or raise alarms in the event of failure. The management infrastructure should support migration of work off of failed servers, which may entail migrating data on local storage or migrating ownership of data on shared storage. An online service provider may choose to offer additional reliability and availability features, in particular, site-wide failover between geographically separated data centers. Such features can offer a competitive advantage over in-house implementations, as well as other online service providers.



It is unfortunate that many techniques for improving scalability have a negative impact on PAR (performance, availability, and/or reliability). Examples of this problem include:

- Keeping session state, such as a shopping cart, in memory on application servers reduces reliability.
- Best-effort caching of data, such as a product catalog, in memory on application servers reduces data consistency.
- In a Web-based application such as online shopping, it is difficult to ensure transactional integrity across the read, edit, and update of a data item. Instead, data such as inventory levels either is not provided or is read into the browser and submitted back to the database in separate transactions. Either way, data consistency is reduced.
- Session concentration, where many smaller machines in the front end of the data center multiplex socket connections to fewer larger machines in the back end, increases response time.
- In asynchronous communication, one-way request messages produce chronologically decoupled one-way response messages. Since messages are generally transactionally enqueued in both directions to handle failures, this technique increases response time.

Applications with weaker PAR requirements permit the use of these techniques and are therefore easier to make scalable. Since scalability is of paramount importance in the online setting, applications with weaker PAR requirements are more suitable for online implementation. Applications with strong PAR requirements can be realized in the online setting, but only by increasing the complexity and cost of the service provider's data center. In particular, the scalability requirements of an online service can be effectively reduced to their in-house levels by giving each organization its own private machines. This is addressed further in the next section.

Enterprise Software as Service

LEVERAGING ECONOMIES OF SCALE

Since the online service model aggregates many users together, it can leverage economies of scale to reduce costs. At the most basic level, volume pricing on hardware, software, and data-center resources such as space and power helps reduce costs. In addition, the more an online service shares resources among users, the lower the cost to add users and the less unused capacity is required to handle peak loads. Homogeneity of requests is an important leverage point for both volume pricing and resource sharing because it reduces the variety of resources that are required.

An essential aspect of resource sharing is amortization of the cost of IT staff across many users. The service provider should develop an application management infrastructure that optimizes the efficiency of the IT staff. Management operations such as tuning, upgrading, and failure analysis should be automated as much as possible. Homogeneity of requests facilitates these goals by reducing the variety of management operations and the skills that are required to perform them. Optimizing operating efficiency can protect the service provider from product commoditization because it offers a dimension other than features along which to compete, just as it does for hardware manufacturers.

The least cost-effective implementation strategy for an online service is to give each organization its own private machines. This approach forces the service provider to provision each organization's system for its maximum load, resulting in considerable unused capacity in the steady state. Nominally, resources should be organized into farms and virtualization technology should be used to dynamically distribute each organization's processes and data. Organizations can realize further cost savings by sharing processes and storage. Sharing can occur at any or all tiers of the system, from Web servers to application servers to database servers.

Within a database, sharing can occur at several levels. The most common approach is to give each organization its own database instance on a shared database server. A more radical approach is to allow each table in a shared database instance to contain data from multiple organizations. Table-level sharing can simplify application management by allowing cross-organization operations such as upgrades to be performed entirely from within SQL. In contrast, with server-level sharing, such operations must be performed on each instance individually and tied together using scripts. If the database provides clustering, then it may be possible to scale table-level sharing to whatever levels the service requires. Otherwise, the system may require multiple database servers and, thus, some cross-instance management. In the latter case, it becomes possible to organize the instances around natural administrative boundaries, such as geographic regions.

One disadvantage of sharing is that it decreases the service provider's ability to configure the system differently for different organizations. For example, if there is a single pool of application servers, then the servers cannot be tuned differently to handle the loads of different organizations. The negative effects of sharing on configuration are even more dramatic within a database, where certain features may become unusable. For example, with table-level sharing, setting up triggers or materialized views may not be feasible for individual organizations. If such functionality is required, the service provider may have to implement it above the database as part of the application.

In addition to enforcing application-level security, such as tracking roles and enforcing access privileges, a service provider must ensure that organizations cannot access each other's data. Sharing increases this challenge. For example, if the application servers maintain data between requests, either as session state or cached database data, then spoofing attacks or buffer overruns could expose one organization's data to another. Similarly, with table-level sharing, bugs in the application or database code are more likely to expose one organization's data to another.

Regardless of the implementation setting, an appli-

cation must prevent individual users from consuming too many resources, either inadvertently or maliciously. Sharing raises the stakes on this issue because it allows one organization to affect another. The basic techniques available to handle this problem are throttling, where requests are delayed so resource usage is spread out over time, and denial, where requests are rejected. Both of these techniques are unattractive in the online setting, where the service is ostensibly being offered as a utility that is available to meet all demands. The service provider must walk a fine line between allowing valid but high resource usage and preventing inadvertent or malicious excess resource usage.

The ability of a user to consume resources, to either a normal or excessive degree, depends upon the kinds of requests the application supports. This issue is affected by a user's ability to customize or extend the application.

APPLICATION CUSTOMIZATION AND ONLINE PROGRAMMING ENVIRONMENTS

Different applications require different degrees of configuration and customization. Simple desktop applications such as spreadsheets are generally configured by the end user though a series of menus or a wizard. Complex enterprise applications such as ERP systems require system integrators or in-house developers to write organization-specific code. At the far end of the spectrum, application servers are platforms for building arbitrary distributed applications. These systems offer different degrees of expressive power to the user/programmer.

In the online setting, implementing highly expressive constructs can be challenging because it is hard to control their resource usage. For example, a simple data-browsing application might offer a fixed set of parameterized queries that are invoked from Web-based forms. A service provider can analyze these queries in advance and accurately predict the load they will place on the database. A more complex reporting application might generate queries containing selections and projections but no joins. A service provider can limit the size or complexity of such queries to help control their running time. In general, highly expressive constructs such as unrestricted queries or arbitrary scripts can be supported only if the management infrastructure can identify and abort operations that are using too many resources.

Highly expressive constructs can make it harder for a provider to upgrade a service. As a baseline, consider a service where the customer provides only raw data, such as account numbers and customer names, with a fixed schema. In this case, the provider can relatively easily upgrade the service without changing the schema or update the schema and do a one-off conversion of the data. As a service becomes increasingly customizable, however, the schema and the internal representation of the customizations may become more dynamic and more complex. If scripting is allowed, support for versioning of the programmatic APIs must be provided.

Highly expressive constructs can also make it harder for organizations to set up and migrate away from a service. For the baseline service just described, it is easy to bulk upload and download the data as a comma-separated-value file or, even better, in an industry-standard format. As a service becomes increasingly customizable, however, it requires more effort up front to get started, and this effort is of value only as long as the organization stays with the service. While such vendor lock-in may be beneficial to the service provider in the long term, in the short term it can prevent the service from reaching critical mass.

Applications with simple, natural customization/programming models are an easier sell in the online setting. If extensive customization/programming is to be supported, then it may well be best to do it through third-party developers. To host applications written by third parties, the service provider must build out additional infrastructure (e.g., to support testing and upgrades). Alternatively, third parties can host auxiliary applications themselves and integrate with the main service using Web services.

WEB SERVICES AND APPLICATION INTEGRATION

Online services are increasingly using Web services to integrate with external systems. These may be existing systems or they may be specifically written to extend the online service. An external system may have its own servers; examples include legacy applications in a corporate data center and other online services. In this case, the external system generally maintains the copy of record of certain business data, and Web services are used by the

online service to extract or input that data. Alternatively, an external system may be a client, either a standard desktop application such as a spreadsheet or a special-purpose rich client. In this case, the online service maintains the copy of the record, and Web services are used by the client to extract or input data.

Different Web services APIs should be provided to end users and third-party developers. End users should be given a simple, strongly typed API that is specific to the data model of their organization. Third-party developers should be given a more powerful, more weakly typed API that spans the data models of all organizations. Note that data models may vary between organizations because of customization. In both cases, bulk operations should be provided to reduce the overhead of network communication.

In the long term, service providers should support Web services APIs for asynchronous communication. Store-and-forward messaging offers a simpler form of reliable communication than distributed transactions and provides a buffer against systems that are overloaded or unavailable. Unfortunately, Web services protocols for asynchronous communication are not yet widely established, and many IT managers are unwilling to open up firewalls for outbound messaging. Thus, in the short term, service providers should support Web services APIs for synchronous call-and-response.

Although Web services add to the power of online services, they do not come without a cost. Unlike the browser interface, a Web services API is neither fixed nor well known. This introduces many of the problems associated with packaged software. A Web services API must be versioned and upgrades must occur without disrupting external systems. Testing and debugging through a Web services API are difficult and require personal interaction between the parties.

EVOLVING A SUCCESSFUL ONLINE SERVICE

To create a successful online service, you should develop a strategy that evolves over time. Start out by targeting an application that is less mission-critical, has weaker data consistency requirements, and requires, at most, simple customization. Focus on developing a resource-sharing strategy that does not jeopardize your ability to support per-organization configuration, enforce security, or prevent denial-of-service attacks. Develop good mechanisms for bulk upload and download of data. Build out a comprehensive Web services API and form partnerships with other online services. Target your service at small to medium-size businesses, which have fewer requirements and a greater need.

Over time, build out a management infrastructure that allows you to deliver the service at the lowest possible cost as a hedge against commoditization. Bring multiple, geographically separated data centers online and offer site-wide failover between them. After you reach a critical mass of users, branch out into additional applications that are more mission-critical and require more customization, and begin to target larger businesses. Q

REFERENCES

1. Ghemawat, S., Gobioff, H., and Shun-Tak Leung, S-T. 2003. The Google file system. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. Bolton Landing, NY.
2. Jacobs, D. 2005. Data management in application servers. In *Readings in Database Systems*, 4th edition, ed. J. M. Hellerstein and M. Stonebraker. MIT Press.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

DEAN JACOBS is a principal architect at salesforce.com. He was one of the original developers of the WebLogic application server and was responsible for WebLogic clustering. He received his Ph.D. in computer science from Cornell University in 1985 and has served on the faculty of the computer science department at the University of Southern California (USC).