Application Service Provider Model: Perspectives and Challenges

Lixin Tao, *Member IEEE* and *ACM*

***Abstract* – With the advance of Internet technology and network bandwidth, Application Service Provider (ASP) mode of computing is becoming a very attractive target of the next wave of Internet revolution. However, the ASP model and its implementation technologies have not been thoroughly investigated.**

In this paper we highlight the essence, benefits, and importance of the ASP model as the first form of commercial service-based computing, and the interplay of the ASP model and component technologies. We survey the main supporting technologies for ASP, identify major challenges to the development of ASP applications, and propose solution approaches. We predict that the ASP model coupled with service standardization will lead to networked economy in which cooperative and specialized computing will be realized at the service level.

***Index Terms* – application service providers, distributed components, Internet computing, distributed computing**

I. Introduction

Computer hardware produces computing *resources* in forms including processor time, data, and data access bandwidth. *Application* software consumes these resources and delivers *services* to users. To users, computers and applications are only means to get computing services.

The separation of resources and application from service can find analogies in public utilities. For electricity, the electrical power is the resource, and light bulbs and electrical heaters, which are examples of electrical application devices, transform the electrical resource into light and heat. Comparably, for telecommunications, the virtual channels are the resource, and telephones and fax machines, which are examples of telecommunications application devices, use this resource to deliver phone-call and faxing services.

Similar analogies can also be made for computing based on the Internet. For instance, Internet bandwidth as well as hardware and data on web and application servers are resources; server software and server-based software embedded in web pages are applications. Users can get services by accessing server data or running the server applications, and using the Internet as an "extension cord" between the server machines and the user I/O devices. Since processor time cannot be easily delivered on a network, the computing applications are usually hosted on servers.

Due to technological limitations, the computing industry has so far mainly adopted the commercial model of selling computers, which are computing resource generators, and licenses of applications. The users are responsible for the installation and maintenance of computing infrastructure and applications, which may constitute significant overhead on the users. For example, such overhead may exclude small and medium-sized enterprises from the benefits of advanced enterprise management services. This model also leads to very low utilization of office and personal computing resources and applications.

On the contrast, the utility industries have long adopted the commercial model of selling resources or services. The resources or services are delivered by comprehensive delivering infrastructures to the users. Users pay by a fixed subscription fee, or by actual usage of the resources or services. For electricity, water, and gas, the corresponding industries sell resources only. For telecommunications industry, in addition to the basic bandwidth resource, *value-added services*, such as directory service, caller-ID service, and call-waiting service, are becoming important components of the services sold to the users. This commercial model enjoys sharing of resources and amortization of infrastructure cost, thus achieving economies of scale; expertise pooling; value-added services; rich experience in service quality level control; rich cost models; and standardization of the resources and main services.

For the computing industry to benefit from the model of selling services, some basic technological and infrastructure hurdles must be overcome. First, the computing resources must be divisible, thus the processor time from a single computer can be shared by large number of users. Second, there should be a comprehensive delivering system for services. Third, the applications must be designed to support server hosting.

The author is with Department of Computer Science, Concordia University, Montreal, Quebec, Canada, and can be reached at taol@acm.org

As early as 1963, with the success of time-sharing operating systems and the emergence of larger computers, a group of engineers from MIT, GE (which later sold its computer department to Honeywell), and Bell Labs proposed to design a new operating system called Multics with the objective of selling computing as services [1]. With the original plan, large computer systems would be connected by telephone wires to terminals in offices and homes throughout a city, and the time-shared operating system would be running continuously with a vast file system of shared programs and data. Even though this objective never materialized, Multics is among the first attempts of computing industry to sell computing as services.

In the last four decades, a series of technological breakthroughs made the dream of selling computing as services closer to reality. Supercomputers and clustering technologies have made huge computing raw power available. Time-sharing operating systems have made computing resources a divisible utility. Personal computers have educated generations of office and house computing users. The Internet has become the largest data and computing service delivery infrastructure, and a new platform of network-centric computing. The World Wide Web has enabled the widespread growth of electronic commerce (e-commerce), and web browsers have become universal graphical user interfaces (GUI) for Internet-based services and thin clients. Component technologies have made it possible to produce huge number of reliable distributed software applications and benefit from specialization and greater economies of scale. Recently, Application Service Providers (ASP) have started a new wave of Internet revolution: use the Internet or other wide area networks (WANs) to provide on-line application services on rental basis. For simplicity, in this paper we use “ASP model” to denote this new service-based computing, and “ASP” the businesses based on the ASP model. While the ASP practices vary from company to company at this immature early stage of the ASP era, we focus on studying the perspectives and challenges of the ASP model of computing in the near future. In ASP terminology, software rental is the same as service sale.

ASP represents the beginning of commercially delivering computing as services. The ASP market is projected to grow rapidly over the next three to five years, from a very small base to over $20 billion in 2003 (Figure 1), yielding compound annual growth in excess of 80% [10]. But for the ASP model to become the mainstream of computing industry, significant breakthroughs have to be made in networking infrastructure, computing technologies, and the rental-based cost models and financial services.

While ASP is changing the infrastructure and model of computing, the ASP model compounded with the effect of distributed component technologies will start a new era of competitive network based computing. The standardization of distributed components will make integration of applications with distributed components a common practice, and the standardization of common application data formats or application programming interfaces (APIs) will further break the monopoly of application service providers, thus instill new energy into the competitive network-centric computing platform. A networked economy will be possible in which modular services from different providers can be easily integrated into new services.

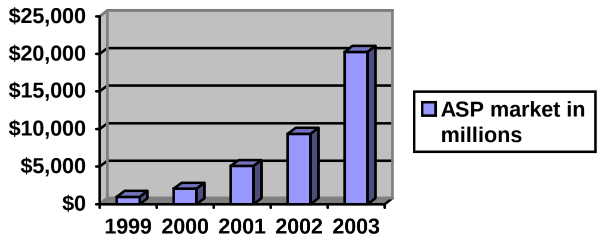

Figure 1 ASP market forecast

So far the ASP model has been limited as a hot subject of industries only, and the research community has not treated the subject at a level commensurate to its importance. This paper aims at surveying the relevant ASP supporting technologies, highlighting the technological challenges introduced by the ASP model, and studying the impact of the ASP model on the general computing infrastructure.

In Section II, we outline the current ASP industry: its identity, value proposition, composition, and services. In Section III, we summarize the current ASP enabling technologies and development platforms. Section IV describes the main technological challenges to supporting ASP. Section V explains why the ASP model will become a corner stone of the new competitive network and component-based computing infrastructure, and foundation of the networked economy.

II. Application Service Providers in Practice

A. *Origins of Application Service Providers*

ASPs are emerging from three separate trends [8][9][10] (Figure 2). From the Information Technology (IT) services industry comes a trend towards selective outsourcing. Among Internet Service Providers (ISPs) the relevant trend has been towards application hosting. Finally, Internet-based enterprises have begun to offer online applications as part of a phenomenon called browser-based computing.

1) *Selective outsourcing*

The practice of outsourcing has a long history in the IT services industry. In recent years, it has evolved to provide increasing levels of granularity in the choices offered to customers. Instead of handing over their complete IT infrastructure to an outside provider, organizations have selectively outsourced specific parts of IT, ranging from data networking all the way through to application management. This has been combined with a trend towards fixed and per-user pricing, often levied in the form of a monthly subscription. ASP propositions emerging out of this strand of development take several distinct forms: application outsourcing, systems management outsourcing, infrastructure outsourcing, whole-environment outsourcing, and subscription computing.

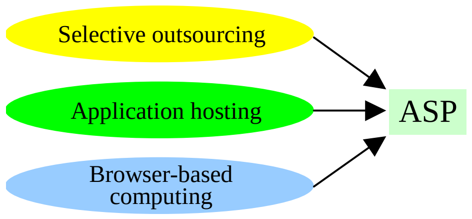

Figure 2 The ingredients of ASP

2) *Application hosting*

Internet service providers have always been ASPs to the extent that the provision of hosted mail and web servers is an application service. Over time, the ISP industry has divided between those who provide access and connectivity services, and those who offer hosting services. The latter, particularly as they move into sophisticated e-commerce, messaging, and other complex web hosting services, are effectively ASPs. A new class of application software vendors, which uses the hosting model to provide Internet-based applications and services, has joined them. Although the Internet industry uses the term *application hosting*, it differs only in name from other forms of application services. There are several distinct subcategories: Internet web server hosting, application server hosting, e-business services, Internet infrastructure services

3) *Browser-based computing*

Web sites started out as Internet destinations that offered only static content, mainly words and images. Today, visitors are increasingly using applications rather than simply viewing content. For gaining the “stickiness” that keeps users returning to their site, information sites have added applications to create dynamic and interactive experiences. Meanwhile, a new generation of software vendors is bringing their applications to market as web-based services, accessed directly over the Internet.

These separate trends converge in serving either specialized business needs or vertical industry markets. This is the advent of browser-based computing – the provision of sophisticated online applications alongside relevant content from a web site, catering to the specific needs of a special interest group. There are several variations, each of them a significant category in their own right: network-based application vendors, Internet business services, vertical industry web sites, Internet marketplaces, and enterprise extranets

B. *Benefits of the ASP model*

The benefits of the ASP model derive from the fact that software applications being distributed over multiple servers rather than dispersed over multiple clients [8][9][10]. The greatest benefits can be derived from the combination of a rental commercial model, a component based application architecture, and a server-based thin-client computing environment.

The benefits for software vendors and service providers include

- *No distribution costs*. Vendors do not have to print manuals, press disks or order thousands of colorful cardboard boxes. They do not have to warehouse, manage, and distribute stock, or operate a returns procedure.
- *No user installation*. Users do not swamp support for help with installation because there is no installation.
- *Fewer illicit copying*. Users usually do not download the software, so they cannot copy it.
- *Instant upgrades*. Suppliers are free to implement bug fixes and new features without having to trace and notify users and then wait for them to download and install the new code.
- *Consistent user base*. The proliferation of different versions and release levels of the software in the user base is either greatly reduced or completely eliminated.
- *Usage monitoring*. Suppliers can monitor usage to gain a much greater understanding of user interaction with the product. They can discover which features are most and least popular, which appear to cause most problems, which need to be streamlined to improve productivity.
- *Potential constant revenue stream*. Suppliers are not obliged to come out with new releases boasting extra features every year in order to maintain a revenue stream. They can achieve the same objective by maintaining a stable user base with a competitive service-quality /cost ratio.

The benefits for users include

- *Limitless choice*. The Internet gives users access to every rentable application that is available online. The model is as persuasive as the PC. Software rental gives them unbounded potential to access and combine services at will.
- *No install hassle*. As soon as the user signs up, the software is ready to use. There is no client installation. The only reason for delay is to have the software configured to meet specific requirements or to allow for integration with other systems.
- *No compatibility issues*. Users do not have to worry about whether their system is powerful enough to run online software or whether the software will conflict with other applications they already have installed,

because they do not install it. They just access it online.

- *No support overhead*. Users do not have to employ expensive administration and support staff to operate complex software installations and the equipments required to run them. The service provider takes care of all that, and averages the cost across all its customers.
- *Reduced downtime*. Most online service providers do a much better job of ensuring 24x7 availability of applications than their customers would be capable of achieving on their own, since it is the providers' core bread-and-butter business activity.

Figure 3 explains why the current ASP customers adopt this new model of computing.

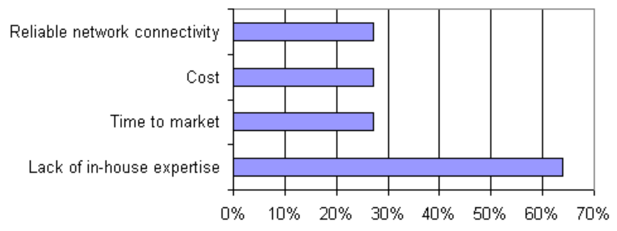

Figure 3 Reasons for users to adopt ASP

The benefits for investors include

- *A high percentage of predictable, recurring revenue through competitive ASP operations*. In a mature market, virtually all of an ASP's revenue should be recurring.
- *Leverage generated from a one-to-many solution model*. This can come from reusable solutions and higher staff productivity due to lower travel rate and higher familiarity with the infrastructure.
- *High switching costs for customers in an immature ASP market*. At this early stage of ASP, many service providers are taking advantage of the lack of service standardization to keep the clients against their wills.
- *High returns on investment through leveraging fixed costs*.
- *An expansion of the addressable market for information technology*. Because of their potentially lower price points, ASPs have the opportunity to expand the market for packaged software to small and middle-sized enterprises.
- *Ability to sell high-margin, value-added services into the customer base*. ASPs will have a highly captive customer base to sell additional services.

The disadvantages of ASP include its difficulty in securing client data and limited performance due to the limited bandwidth of the Internet. We will address these issues in the later sections.

C. *ASP Channel Stratification*

There has been a stratification of the ASP channel into a number of interlocking layers, each with its own areas of core competence [8][9].

While an end user customer experiencing an ASP solution will deal with only one provider, in most cases that solution will be made up of various components coming from several different layers of providers. Among those hidden layers there might be a company that, even though it is a major contributor within the solution, never has a direct ASP relationship with the end customer.

This stratification is a natural consequence of the multi-tiered computing architecture that enables the ASP model. With various elements of the solution performed on separate, specialized servers, it becomes an obvious next step to have each of those elements catered for by separate and specialized providers.

Some ASPs continue to argue in favor of a vertically integrated model in which they own and control every element from top to bottom, while others promote the merits of outsourcing to best-of-breed providers. The former approach can deliver tighter integration and more assured control, while the latter normally benefits from greater economies of scale. But every provider outsources at least some element of the solution. Those who host at their own server centers rarely write their own software. It is up to providers - and their customers - to weigh the risk of outsourcing against the cost of in-house provision, and strike the balance that best suits their particular requirements.

The four primary subdivisions of the ASP channel are set out below (Figure 4). Within each of those layers there are many different possible components. An ASP solution might be made up of contributions from a dozen or more different providers, each responsible for just one component. Alternatively, just one or two providers may fulfill it, each of whose activities straddles several layers.

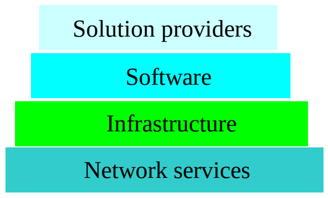

Figure 4 ASP channel stratification

1) *Network services*

At the network layer sit the providers of basic communications, server center resources, and value-added IP (Internet Protocol) services. Communications include the physical connections, the routers that handle the IP traffic, and the associated performance, reliability, and security applications. Server center resources typically embrace the provision of collocation space, protected electricity supplies, and physical security and maintenance services. Value-added IP services include virtual private networking (VPN), network caching, streaming media, firewalls, and directory services.

2) *Infrastructure*

The next layer is a rapidly emerging space with rich pickings for talented early entrants. Many providers offer individual services such as utility storage and server hosting, or operational resources such as call centers, finance, technical support and so on. Some have gone on to coordinate third-party services along with their own in-house skills and resources to provide a complete infrastructure that allows their clients to operate as ASPs. This ASP infrastructure provider (AIP) role includes the coordination of network and systems management, the supply, operation and management of systems hardware and software, and the management of ASP subscriber accounts, billing and customer support. A further important element comprises application management, service level monitoring, helpdesk infrastructure, and the streamlined messaging of alerts and support information between partners within the ASP channel stack.

Many ASP pioneers have gravitated towards this AIP role, sensing the opportunity to turn their early experience into a marketable commodity that can be packaged and sold as a solution to newcomers. They offer the service to independent software vendors and systems integrators who wish to bring existing client/server applications across to the ASP environment, often advising on the fine-tuning and re-engineering required to enable them to run effectively from a shared, Internet-based server center.

3) *Software*

Software providers add the vital ingredient that enables the finished application service. The software may be a ready-made, packaged application that is adapted for ASP delivery, or it may be specifically developed for the purpose. There are a number of application server platforms suitable for the creation of ASP offerings, although at present few offer a complete set of services for functions such as service deployment, subscriber management, support, service level management, and billing.

While most development is done in-house by independent software vendors (ISVs), a growing number of software companies and systems integrators are developing specialized skills in building online application services to order.

4) *Solution providers*

Solution providers fulfill the final step in the chain. These are the true ASPs, who package the software and infrastructure ingredients together with business and professional services to create a complete service product to present to the end customers.

III. SUPPORTING TECHNOLOGIES FOR ASP APPLICATIONS

Applications running on ASP servers need special properties, such as separation of business logic from presentation with Internet protocols; reentrant code; scalability; efficient storage and retrieval of session data; and lifetime management.

Most existing client/server applications are not suitable for ASP hosting. But for those who believe time-to-market is more important than quality of service, there are some technologies to support fast-track adaptation of existing client/server applications for ASP hosting by logically extending the cord between ASP servers and client PCs' I/O devices.

Microsoft Windows 2000 Terminal Services (WTS) is an example technology in this category. WTS allows standard Windows-based client/server applications to run on the server instead of on a client PC. Clients running Windows terminal software can then access the sessions through Microsoft Remote Display Protocol (RDP).

As another example in this category, Citrix, the original developer of the core technology underlying WTS, has its own Independent Computing Architecture (ICA) for delivering sessions to clients, which supports non-Windows clients on platforms such as Java and Unix as well as the Windows clients supported by Microsoft RDP. It also offers a technology called Application Launching and Embedding (ALE), which allows Windows applications on the server to be accessed from any browser without the need to install special Citrix or Microsoft client software.

To fully benefit from the ASP infrastructure, ASP applications must be designed and implemented for ASP purpose. The complexity and cost of such applications mandate their adoption of the component approach. Even though different ASP applications may provide different services, they do share many functions like user subscription and management, billing and payment processing, service quality control, data storage and management, and authentication and verification. The rule of thumb for the success of a small or medium-sized enterprise is: spending 90% of its investment in optimizing the 5-10% components in its expertise domain, and adopting commercial-off-the-shelf (COTS) components for the rest of its applications. For advanced external components it may be better off to let them be hosted by other specialized ASPs, which is exactly the concept of *service integration*.

A common concern for the component approach is the quality of external components, since the source code of them is usually kept away from the system integrators. Our arguments are: in-house code is usually inferior to components implemented by specialists; market competition is the major driving force for producers of components based on publicly-accepted standard APIs to perfect their products; and public adoption of a commercial component will lead to early discovery of its bugs and deficiencies.

A. A Reference Model for Distributed Components

A component is a binary module of code that supports system integration. A component is usually implemented as an object with some extra properties. A distributed component further supports interoperation and collaboration of components running on different processors across a network. A full-fledged distributed component usually has the following properties:

- *Universal reference*. Each instance of the component must have a reference or ID unique across the world, so that other component instances can call methods on it through the Internet. Such reference should be valid across various computing platforms, component implementation languages, network protocols, and geographical distances.
- *Network interoperability*. Any two component instances on the Internet should be able to interact with each other without regard to the computing platforms, implementation languages, network protocols, and geographical distances.
- *Introspection*. Without the source code of a component, the computing environment or other component instances should be able to dynamically find out the API of the component including types and signatures of public attributes and methods. This will enable dynamic interaction between two component instances unknown to each other.
- *Customizability*. The attributes and behavior of a component instance should be able to be customized off-line, usually visually with an integrated development environment (IDE) tool, without touching the source or binary code of the instantiating component.
- *Toolability*. The above customization, as well as component integration, should be able to be carried out in a visual tool environment. This will enable system integration and management without coding.

Since the component instances will interact and collaborate across networks, a component supporting system must be used to provide various services either on-line or off-line. Typical services in this category include

- *Naming*. It associates user-friendly names with references of component instances. The names should be globally unique.
- *Trading*. Just like phone yellow pages, trading service will allow new components to be published on the Internet, and looked up by potential users according to categories of services.
- *Life cycle*. Life cycle service is responsible for handling the instantiation, migration, copying, and destruction of component instances. It is the key to support visual dynamic integration of applications and services.
- *Persistence*. Persistence service will allow component instances to be activated automatically upon client invocation; it will automatically save the state of a component instance during server crash, restore the saved state upon re-instantiation, and support the illusion that component instances and the references to them are persistent.
- *Event*. This service is the key to support event-driven execution among components. Event services are usually components themselves. Event source and sink components can register with an event service component. A sink component can register itself as either a “push” or a “pull” client for a particular category. Upon notification from the event source, the event service component will broadcast the event to all of its registered “push” clients for a particular category. The “pull” clients can check out the events with the event service component at their own timing. Event service is also a convenient tool to support the “push” and “pull” of information.
- *Transaction*. Transaction service makes a sequence of interactions among components atomic: either they all succeed, or none of them will commit any state change for the relevant components. Since such transactions are not specified in source code, transaction service can support dynamic or integration-tool based specification of transactions, or transactions among components implemented in different languages.

B. Major Component Technologies

In the following Subsections we describe briefly three major distributed component models based on our component reference model. They represent the latest industry technologies that support ASP server applications.

1) CORBA

Common Object Request Broker Architecture (CORBA) [4] is the dominant distributed component model for ASP applications for which the components need to be deployed across various types of networks and on various platforms. The Object Management Group (OMG), an industry consortium consisting of over 800 IT companies, with the noticeable exception of Microsoft, specified CORBA.

CORBA uses Object Request Broker (ORB) to provide network connectivity for its components. It uses a neutral

Interface Definition Language (IDL) to separate interface specification from the implementation of a component. CORBA components support all of our component properties except customizability and toolability. Currently all Netscape web browsers have a built-in ORB to support CORBA based applications embedded in web contents.

The component management services for CORBA components are supported by CORBAservices, which covers all of our listed supporting services, and much more.

A special feature of CORBA is that it can easily wrap up legacy code in CORBA wrapper components, thus providing a fast-track approach to adapt legacy code to the ASP model.

The ultimate goal of CORBA is system integration. OMG uses IDL to standardize the specification of vertical and horizontal common facilities for system integration.

2) *Enterprise JavaBean (EJB)*

Java, developed by Sun Microsystems, has become the foundation of a powerful environment for developing and running server-based applications [3]. Enterprise JavaBean (EJB) is a component-based infrastructure framework that forms the basis of many high-end application servers. While JavaBean is the model for Java components that mainly run on a client machine, an EJB is a specialized, non-visual JavaBean that runs on a server. The EJB server-side component architecture brings together most of the properties and services we described for our component reference model.

EJB by itself can only support the integration of Java components. But EJB and CORBA are complementary. CORBA has become the implementation technique of EJB Remote Method Invocations (RMI). EJB has provided CORBA with a user-friendlier user interface. EJB augments CORBA with declarative transactions, a server-side component framework, and tool-oriented deployment and security descriptors. CORBA augments EJB with a distributed object framework, multilingual client support, and IIOP (Internet-Inter-ORB Protocol) interoperability.

To integrate an existing CORBA component into an EJB framework, we only need to use the IDL-generated JavaBean proxy to represent the original CORBA component. Therefore it is very easy to take advantage of both of these two component models.

3) *Microsoft DNA*

While CORBA and EJB are both based on open standards, Microsoft Distributed interNetwork Applications (DNA) is a proprietary technology based on Microsoft's COM+ [6]. DNA represents Microsoft's vision of networked computing; it is an application architecture to compete with CORBA and EJB. The objective of DNA is to fully embrace and integrate the Internet, client/server, and PC models of computing to support the development of scalable, multi-tier business applications that can be delivered over any network.

The core of DNA is COM+, which is an enhanced version of (Distributed) Component Object Model (COM/DCOM) integrated with Microsoft Transaction Server (MTS), the supporting environment for COM components. Unlike CORBA and EJB, COM is a binary standard to support the interaction among component instances with pointers to interfaces.

The main strengths of COM+ include its position as a mature core supporting technology for Windows applications in the last decade; close integration with Windows applications; and user-friendly development environments. Therefore it is a very attractive platform for developing ASP applications for servers and clients using Windows. The disadvantages of COM+ include that it is basically native to Windows platform; its limited services for distributed computing and limited scalability at this time compared with CORBA; and the lack of competition to perfect the technology. Even though Microsoft has made great effort to port COM to other platforms, MTS, the supporting environment for COM, proved to be a big obstacle to this effort due to its high platform dependency. In recent years Microsoft tried to use interoperability to compensate COM's platform dependency. With DCOM/CORBA bridges, COM component instances running on Windows can interact with CORBA/EJB component instances running on other platforms. Microsoft supports Java, but only as a language, not as a platform.

IV. ASP CHALLENGES

At this time ASP is still in its stage of proof-of-concept. To truly benefit from the ASP model of computing, we have to address many challenges in technology, financial infrastructure, and law. In this Section we only discuss some of them related to computing technologies.

A. *Scalability of ASP Servers*

Application hosting servers need to support tens of thousands of concurrent service sessions with high availability and short response delay. Not like web servers that are mainly used to support stateless HTTP connections requesting web contents, ASP application servers need to support connection sessions during which some state (session data) must be kept on the servers. Such sessions may last hours or days, and servers cannot predict the connection patterns.

For web servers, the current main techniques to improve server performance include RAID disk array; server farms based on dozens of processors interconnected by buses or shared memories; and extensive caching. For example, Yahoo uses an array of around 50 high-performance server processors, and Windows 2000 can support a limited number of server processors. Non-preemptive scheduling algorithms are used to balance the workload among the processors. Some application service providers also support external caching as a generic approach to boosting web server performance. Today's web server market is mainly based on proprietary ad hoc techniques, which cannot support the level of service quality needed by ASP.

Due to the need to support session data, caching will be much less effective for ASP servers. Dozens of processors may not be enough to support the data processing power of a full-fledged ASP server. The current bus or shared memory based architectures introduce bottlenecks in server performance.

To achieve the scalability required by ASP servers, we need to investigate the preemptive process or object scheduling techniques [2] in the environment of ASP servers. A server will consist of one or more master processors and a cluster of client processors. The master processors maintain dynamic load information of the client processors. The client service request will first reach one of the master processors, which will become the gateway for further communication between this client and its client processors. The master processor will start a server process on a client processor based on the current load distribution of the processors, the level of service quality required by the client, and the scheduling policies. Since the clients use the ASP servers in unpredictable patterns, the initially lightly loaded client processor may become heavily loaded afterwards. With preemptive scheduling, a process can migrate from one processor to another based on particular scheduling policies to rebalance the workload. The major challenges here include how to minimize the process migration overhead; how to share and maintain client state (session data); and how to reroute the communications between the gateway and the client processors with minimal footprint in the client processors and minimal overhead.

B. *Internet Infrastructure*

Distributed components and services mainly use or will use URL addresses to uniquely identify each other. Currently the Internet mainly uses the 32-bit Internet Protocol (IP) addresses specified by IP version 4 (Ipv4). But we are running out of address space in the Ipv4 address base by 2005. We need to have IP version 6 (Ipv6) in place by then. Ipv6 uses 128 bits to represent a URL address, and allows ample expansion space for future component-based ASP services. Windows 2000 already supports Ipv6.

With ever-increasing communications volume on the Internet, the response time of ASP services depends on the speed by which the information flows between clients' Web browsers and the ASP server centers. Data communications are mainly supported on the Internet by packet switching today. With the achievable bandwidth of 1 Tb/s (10^{12} b/s) per optical fiber, the communications delay on the Internet will be mainly dominated by the number of hops from source to destination, not by the bandwidth of individual carriers, which are coupled by routers. When a packet arrives at a router, the packet needs to be buffered, its header needs to be extracted for destination address, and a routing table or algorithm will be used to determine its outgoing channel. A typical message will be cut into many small fixed-size packets, thus incur significant overhead in routers along the way to the destination.

One possible approach to reduce the delay of router processing is to adapt the wormhole routing [5] of parallel computing to the Internet. With this approach, a message is cut into packets, and each packet is further cut into smaller units named flits (flow control digits) the bits of which can traverse the communications carrier in parallel. For a packet, the first flit contains the destination address, and the last flit signifies the end of a packet. When the first flit arrives at a router, the router hardware will set up a passage to bypass the incoming channel to the out-going channel based on the destination address and the routing algorithm. The following flits can just bypass the router without being buffered or processed. When the last flit arrives, the hardware passage between the incoming and out-going channels will be broken up, and the router resources can be recycled. With this approach, if the network is not congested, and the number of flits for a packet is much larger than the number of hops that the packet needs to traverse, the delay for the packet is roughly proportional to the packet size, not to the hop number. CISCO has implemented a variant of this approach in some of its switches [11]. Based on today's Internet infrastructure, a packet can move around the world in around 14 hops or less.

C. *Micropayment*

For ASPs to fully benefit from the new model of operation, the on-line billing and payment mechanisms must be smooth, secure, and efficient. They should support both very large transactions and very small transactions, the latter may involve only a few dollars that the traditional financial institutions do not process.

Today's dominant mechanism for on-line e-commerce paying is credit cards, which supports neither the very large transactions nor the very small transactions. The major challenge is to design mechanisms that support the collection of very small sums, or micropayments, so that clients can pay-as-you-go and not be deterred by complicated payment overheads. The current approaches under investigation include electronic money, virtual money, digital money, and smart cards [7].

Micropayment is much more complicated when an ASP service is implemented by integration of distributed components from several ASP providers. Such components may be downloaded to client sites during usage for performance or security. In this situation the client payment may need to be distributed transparently to multiple involved service providers based on accurate usage statistics and service contracts. The situation will be easier to handle if all commercial distributed components support standard API and mechanisms to record and maintain usage statistics, or *microaccounting*.

D. Security

This is one of the first issues that potential customers typically raise with application service providers. ASP security addresses both client data security and server availability.

Today's virtual private network (VPN) technology can be used to make any Internet connection highly secure against outside interference. Similarly, it is easy to make direct access, either through dial-up or with a leased line connection, secure using encryption. At the server centers, the use of firewall technology further guards against unauthorized access.

It is much more difficult to establish internal staff procedures that are sufficiently robust to protect against security breaches. The vast majority of security lapses involving information technology today are not caused by hackers breaking through electronic security defenses, but through careless or malicious acts by employees. ASPs will have to establish stringent procedures to ensure that the integrity of customer data is not compromised while it is under their care.

For extremely sensitive data, it is necessary to support the fat mode of ASP: allow clients to download the necessary subset of components to process data on clients' desktops.

ASP server centers must be armed against malicious attacks in forms of viruses and monopoly of communications and server resources. The examples of such attacks in 2000 include the *mafiaboy*'s attack to major web servers in America by exponential number of HTTP page requests that paralyzed the web servers for several hours, and the *love virus* that blocked email services around the world. These attacks could reduce the confidence level of customers to ASP services. At this time, most web servers, including Microsoft IIS (Internet Information Server) and Apache, would crash themselves and the underlying operating systems when the concurrent client connections to them exceed the capacities of their scalability.

E. Dynamic Configuration

In the ASP environment, multiple clients of different configuration requirements may use the same application, and they may run the application concurrently. Therefore, an ASP application must be able to support separate configurations for each independent group of users. This is in stark contrast to current practice in enterprise systems management, where it is the norm to enforce a common standard throughout the enterprise in order to ease systems management complexity. ASPs do not have the option of mandating consistency across the user population, but instead must embrace and manage that complexity. Neither conventional client/server nor next-generation e-commerce application architectures currently have satisfactory answers to this need to support and manage diversity.

F. ASP Service Integration

An enterprise may not want to entrust all its computing needs to a single service provider. First, it may have significant amount of proprietary or legacy systems and applications to run. Second, it may have sensitive data to protect. Third, it would like to benefit from competitions among the service providers for better service quality and lower cost.

On the other hand, few service providers can afford to be fully self-contained. For example, many ASPs will opt to let professional financial institutions or companies run the credit card payment services, so they can reduce the operation cost and increase the confidence level of their clients. There is a strong need for easy integration of existing ASP services to create new services.

Therefore, ASP applications should support the integration of ASP services from different ASP providers, and the integration of ASP services with the client applications. There are two basic approaches to this problem. One is the standardization of common applications' data formats. The other is the standardization of common applications' APIs. They are further explained in Subsection V.A.

V. ASP in Perspective

A. Breaking ASP Monopolies

The ASP model alone can easily lead to monopoly of services. Current ASP market mainly consists of service providers for existing standalone or client/server applications. Applications in the same category usually have similar functionalities, but different data formats or user interfaces. Many data formats, like Microsoft Word, are proprietary and kept from the public. Changing format of user data, while possible in some simpler cases, is problematic in general, especially when a proprietary data format is involved. Clients of such service providers will soon find that it is almost impossible for them to switch to other service providers that may provide similar services but with a different application. While many ASPs today

[10] cite "high switching cost for customers" as a major advantage of the ASP mode of computing, such switching cost will definitely deter the adoption of ASP by many new clients, and impede the competitive innovation thrust in the computing industry in general.

To promote competition in the ASP market, and expand the ASP market with non-committed free trials, public standardization of services must catch up. There are two levels of standardization here. At the lower level is the standardization of data formats of major applications. All ASP services complying with such standards can process user data in such public data format, but users may need to learn the different user interfaces of the alternative services. At the higher level, in addition to the standardization of data formats, the user interfaces of major applications in the same categories are also standardized. Services based on new applications complying with such standards can be easily adopted by clients without a learning curve. Innovative companies can improve the performance of existing applications, and grab the user base from less competitive service providers with higher service quality.

But who should be in charge of such standardization? There are two choices: governments or industry consortiums. Object Management Group (OMG) is an example of such industry consortiums. Individual companies will usually not publicize its proprietary standards. For many companies, the standardization process is not voluntary, but mandated by the market economy.

B. *Network-Centric Computing*

The ASP model is based on the advances in distributed computing in the last two decades. But the ASP model will also promote network-centric distributed computing to a new level of scale.

The essence of distributed computing is to use networks to promote cooperative computing and specialized computing. The Internet expanded such cooperation and specialization to the global scope. Componentization of distributed software made distributed computing more reliable, more convenient, and less expensive. The ASP model promotes cooperative computing by simplifying the client side's computing device and application to a web browser or a thin client, and running the applications by specialized service providers. As a result, the Internet is emerging as a new platform of global computing.

The new generation of distributed computing will be characterized by component-based finer computing granularity; global cooperation and specialization; multimedia data; binary integration; mobile computing; and its omnipresence in electrical/electronic devices.

The application of the above generic network-centric computing will lead to networked economy characterized by integration of services.

Reference

[1] F. J. Corbato and V. A. Vyssotsky, "Introduction and overview of the MULTICS system," *Proceedings of the AFIPS Fall Joint Computer Conference*, 1965, pages 185-196

[2] A. Goscinski, "Distributed operating systems: the logical design," Addison-Wesley, 1991

[3] James Gosling et al., "Java programming language, second edition," Addison Wesley, 1998

[4] Thomas Mowbray and Ron Zahavi, "The essential CORBA: systems integration using distributed objects," Wiley, 1995

[5] L.M. Ni and P.K. McKinley, "A survey of wormhole routing techniques in direct networks," Computer, 1993, pages 62-76

[6] David S. Platt, "Understanding COM+," Microsoft Press, 2000

[7] Mostafa Hashem Sherif, "Protocols for secure electronic commerce," CRC Press, 2000

[8] Phil Wainewright, "Anatomy of an ASP: computing's new genus," *ASP News Review*, www.aspnews.com, January 2000

[9] Phil Wainewright, "Packaged software rental: the net's killer app," *ASP News Review*, www.aspnews.com, January 2000

[10] "ASPs: the net's next killer app," www.aspisland.com, January 2000

[11] "Bridging and switching basics," www.cisco.com, June 1999