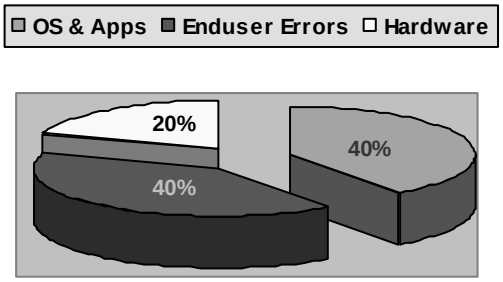
Do Not Ask for Whom the Server Fails:

reliability, availability and web services

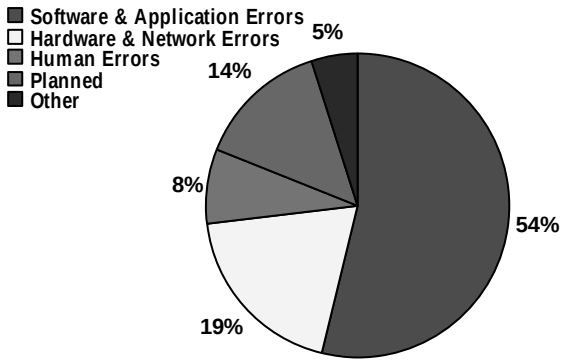
The ability to invoke methods or run procedures on computers physically separated from a user's local machine via some network is not new. This arena of remote computing has been fraught with peril though even when the basic quanta in question has been the entire program to be run. Web services have brought this paradigm of remote computing down to an even finer grain in an attempt to treat these remote programs as if they are integral parts of another program, whether local or remote itself. The reliability and availability of servers is obviously quite important when invoking programs on them remotely and is even more critical when parts of a program make use of web services located on machines anywhere on the planet. If a program that makes use of web services does fail, it would then not be apparent whether it failed due to an internal inconsistency, or whether a part-time computer operator tripped over a cord half a world away, only that the program did indeed fail. The reliability and availability of servers hosting web services is then of utmost importance to the growth and use of web services. Constant availability and perfect reliability are targets that certainly cannot be hit. The question is then can web services be made reliable *enough* so that they are as near to perfect as necessary, or will the reliance on breakable hardware, buggy software and fallible end users assure the stagnation and death of web services?

In their analysis of causes of downtime of web servers, IBM determined the rough percentages depicted in the pie chart below in 2002. [1] Take careful note of the percentages. Apparently excuses on the order of, "The hardware went south" are not acceptable. You have a much better chance of being believed if you lay the blame on the end user or the software. Job security aside, the increase of reliability of all of these parts would allow for a more fertile environment for web services to grow. Here web services are somewhat lucky since it is also quite financially attractive to improve the reliability and availability of servers.



The same IBM study of contemporary sources estimated the average cost of server downtime in lost revenue to be approximately $10,000 a minute on average with rates of as much as $27,000 a minute not unheard of. [1] The same study also found that users often experienced two-thirds more downtime than they had originally expected. [1] Another study done by the Standish Group, a company devoted to developing very accurate pictures of why servers fail, and then marketing really good server middleware, generated the pie chart below when trying to determine the same thing as the IBM study done in 2002. It is rather surprising that two of the three categories listed in the IBM study come in very close to the Standish Group study. The one area of contention, human error, may actually be as important as the IBM study makes it out to be, albeit difficult to ferret out and solve. Bad

hardware is bad hardware. It is easier to test to determine if it is working, and if not, it is often apparent what the failure was. Human error does not possess these nice diagnostic abilities for ease of detection and repair, so human errors could be misinterpreted as failings of other components, especially in light of the career advice given above.



Since the cost of downtime is so high and the expectations of users are obviously higher than the quality of current servers, the market for more dependable and reliable servers, especially those that are tolerant of user mistakes and OS and application errors, should be rather healthy. If more powerful, self-reliant servers have been steadily introduced since the Standish Group and IBM studies from 2002 were published, web services stand a good chance of being accepted by the computing community at large in the future since the platforms they depend on would have been engineered to be as nearly perfectly reliable as possible. Web services would also have a rosy future if the current leaders in technology foresee and are working towards a future where ubiquitous computing will be the status quo with computers connected wirelessly surrounding us wherever we go, just like in all those sci-fi movies that warn against just such a thing. This is in fact the case.

Taking the last example first, last week at the Gartner Symposium/ITxpo 2004, Bill Gates gave his vision of computing in 2014, a future where industry and application logic would be focused on by software companies and coding itself would be abstracted so that they "…could reduce amount the amount of code they need to write by at least a factor of five. In addition, he touted Web services and XML as key to software development trends." [3] Right now Microsoft is sitting at the top of the general computing hill, and it doesn't look like it intends to come down any time soon. If Microsoft will begin to push the web services paradigm of cobbling together useful and powerful applications in very small amounts of time using functions that are readily available as web services, it would be a matter of years before web services would be commonplace. Although Microsoft may really, really want to build applications in a matter of weeks using web services it won't happen if their servers aren't reliable.

So, how close are we now? Apparently, we're well on our way there. IBM offers a plethora of different types of servers, among them are the xSeries and the iSeries. Since IBM's servers are used quite often, there is a relatively plentiful amount of information about these servers. The xSeries are the 'Cadillacs' of IBM's servers and have a wide set of features built into them to provide increased reliability, one of them is a relatively reliable memory subsystem. The OnForever™ memory system uses memory mirroring so that the loss of nay single DIMM will not cause the server to actually lose the data on it and can correct single-bit errors. [4] The same feature set also allows for the removal and replacement of memory chips while the system is running. [4] So, if a DIMM dies, a web service the server hosts may slow down, but will certainly not come to a screeching halt. For that matter, the ability to add and remove PCI cards, additional servers and other resources can be done while the system is up and running.

The xSeries also have an integrated computer that monitors critical system components and conditions and can engage in 'proactive systems management', sending warnings when the hardware is starting to near the end of its life cycle, generating errors, performing poorly, or if system conditions are exceeding tolerances. [4] IBM avers that hardware failures can be predicted up to 48 hours before they occur, allowing proactive hot-swapping or purchasing or locating of spare parts. The more advanced versions of the system manager can actually monitor the use of hardware by software and can detect when software claims resources, but does not release them. Incidences of these claimings of unused resources often increase over time until the software in question eventually fails. [5] For those of you who are wondering, IBM never states as much, but I am sure they have learned their lesson from *2001: A Space Odyssey* and have made sure that their proactive systems managers will not suffer neuroses, sense that a component will fail, and then kill all the astronauts on board.

If such a reliable system can be built, why are web services not already exploding through the industry? There is, first and foremost, the issue of cost. Big servers are expensive. Big servers with nice features are even more expensive. Are these servers actually worth it? Apparently they are. Early last year IBM had retooled some of its products and sent them to International Data Corp (IDC) along with some case studies for analysis. [6] All of the case studies involved mid-range companies, the six that IDC selected averaged 725 employees, with an average IT staff size of 11. [6] IDC then normalized its findings per 100 users in order to make the data a bit more relevant for smaller companies. [6] The major finding was that with an average initial investment of "… $141,227 per 100 users, companies could save an average of $542,728 over the course of three years." [6] The savings came from consolidating multiple machines into single servers using IBM's IxS and IxA cards.

IBM's IxS card is basically a single board computer (SBC) that can use the devices attached to the server as if they were its own. A SBC is a motherboard with all of the necessary parts embedded on it, reducing the size, power consumption and price dramatically. IBM's IxS cards view virtualizations of partitions on the hard drives in the server the IxS is slotted into as its own physical drives, so it needs none of its own. IxA cards are even more esoteric, allowing IBM xSeries servers to connect to iSeries servers, which would then view these external servers as if they were SBCs. What this allows is centralization of some of the IBM reliability features and the expanded use of physical devices, although more importantly it allows companies to buy a cheap SBC and slot it into an existing server that has provisions for sensing and working with this hardware quickly and easily rather than buying a new machine and trying to integrate it into the existing network. Different operating systems and applications can be run on the different SBCs, since they are, in effect, completely different machines connected by a virtual LAN internal to the iSeries server they are plugged into. [6]

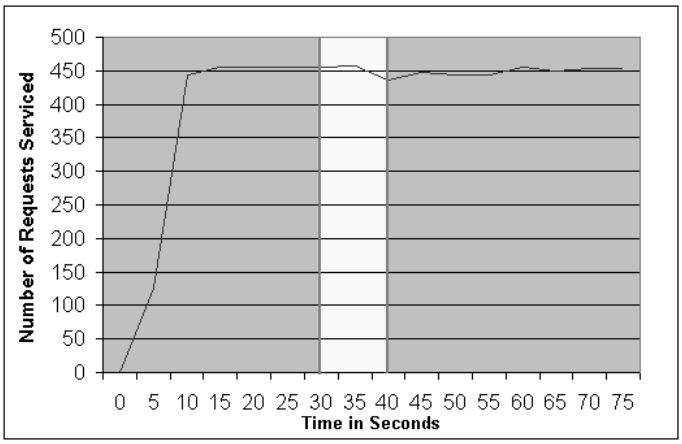
The savings found by the IDC were basically due to the ability to consolidate multiple machines into a single server, making administration easier, although the additional reliability added by the IBM servers' hardware was the dominating factor. [6] Before consolidating their machines and functions into IBM servers with IxS and IxA cards, the actual normalized average downtime experienced per 100 users was estimated to cost $528,393, the increase in reliability after migrating to the IBM hardware caused the amount of lost revenue due to downtime over the same amount of time to drop to a

normalized average of $49,779 per 100 users. [6] This came from only a .125 % increase in reliability. Apparently the financial side is all but a lock. Even though the difference in reliabilities was an increase of only .125%, the actual percentage of uptime was a startling 99.987%! [6] Apparently basic hardware reliability is all but perfect, as long as you can afford the servers, although with a normalized average decrease of 10.6 to 1 in the costs of downtime achieved by migrating to better hardware, the servers may very well pay for themselves. [6] Reliability needed to be increased a relatively small amount in order to achieve significant monetary savings and is already reliable 99.9+% of the time, but the reliability may still not be seen as being high enough for hosting web services due to other factors.

Although important, perhaps the reliability of the hardware itself is not the determining metric, but the reliability of the overall architecture as well. Several different server architectures have been implemented and are being researched in order to increase server reliability. One such feature in production currently is the ability of IBM servers to restart themselves if there is a critical problem with the server that requires a restart and no human is there to do so. [5] This hardware could be used to implement a highly-reliable SSM system that operates by tracking user session states and rebooting automatically on failure. In fact the prototype SSM system was developed on an IBM iSeries server cluster at Stanford University. [16]

The SSM architecture sets up housekeeping in the uncomfortable middle ground between state-full and state-less web services by compromising instead on a ‘session state’. SSM writes information to a set of distributed ‘bricks’ and on a read needs to have some number of these remote ‘bricks’ agree in order to regard the data as correct. [16] The individual server ‘bricks’ then signal the application servers with a heartbeat, so a ‘brick’ being restarted can quickly reinsert itself back into the system and have the newest data sent to it. [16] Not only could the IBM servers restart themselves on hardware failures, but could also restart server nodes running SSM software if a program on the node begins misbehaving. SSM does in fact record user state, so it is not state-less, but a server node can be lost and then later rejoin the group and function as if it never left, so it is not entirely state-full.

The chart below shows the effects of one ‘brick’ of four dying 30 seconds into the simulation while requests are being serviced. There is barely any change in response time, and no requests are dropped. [16] This seems then to be a model of near to absolute reliability. Not only can the individual servers monitor their own hardware and even predict when it might fail, allowing it to be replaced on the fly when it does, but can monitor their SSM programs to determine if they are malfunctioning, and if so, restart themselves. As long as a



large enough cluster of these servers using the SSM architecture is available, the services they support would be all but entirely reliable.

SSM requires more than just one or two ‘bricks’ to be useful, and that completely disregards the cost of application servers utilizing the ‘bricks’, DMZ servers, web servers, and any other necessary infrastructure. SSM also begins to rather liberally reject client requests for service when the ‘bricks’ begin to become overloaded as a form of backpressure. [16] So, the more demand there is for a specific service or set of information, the more autistic SSM will become. This would decrease the system’s availability, albeit decreasing the load on the ‘bricks’.

There are also new types of hardware failover techniques being developed. Once such technique is the use of Backdoors, being developed at Rutgers University. Backdoors is a novel solution to the phenomenon of hardware, application, or operating system failure. The reasoning being that if an application causes a critical failure, or an operating system dies, the computer in question still retains the data it was working on in memory, which will still be powered and accessible. [7] While operating all nodes have their OS subsystems update a progress box (PB) that is located in protected system memory. Every so often the nodes will query the nodes they are monitoring over an internal network (although in order to do this the nodes’ OSes need to support remote access). [7] If the PB has not been updated after a certain amount of time, it is assumed that the node has failed. [7] While the nodes service requests they store user state information in state boxes (SBs) in memory. [7] When a node is considered dead, the node (or nodes) monitoring it read the SBs from the failed node and take over its IP, in effect transferring the pending transactions from the failed node and taking over any future load that would be sent to it. [7]

So what’s the catch? Apparently there is none. The entire process happens rather quickly and is entirely transparent to the user. While operating under a load of 100 clients, serving about 700 requests per second, when a node died the total observable delay was about a quarter to half a second. [7, 8] The monitoring overhead can be made to be rather low, yet still effective. Overhead can be brought down to 1% of the CPU time if done at 100ms intervals on relatively powerful servers, 2.4GHz dual processor DELLs.[7] The occurrence of false positives at a monitoring loop of 100ms has been derived and tested to be 0%, so nodes should not annex the load of healthy nodes. [7] As of yet security has not been dealt with, but so far the results are quite promising.

Although these new technologies are very powerful in the own right, if they can be refined and combined with each other server reliability can be increased even more. A Backdoors-type monitor imbued with SSM-like recovery qualities on a server that allows the hardware to restart itself would engender a server with a new level of self-healing. In the event of a fatal software error on one of the server’s nodes, the client state on the injured node would be transferred to another node, its IP and load taken over via the auspices of Backdoors. The node could then be restarted by a SSM-like mechanism working with server hardware that allows self-restarting. Once the node was back up it could signal the rest of the nodes, as in SSM, and have its IP and load restored by the node (or nodes) that picked up the slack in a reverse of the Backdoors recovery process. All this would be handled auto-magically by the sever and would be entirely transparent to the user except for a small hiccup, a quarter second or so, in response time at failure, and slightly degraded service until the restarted node finishes booting. These new

technologies are still in the prototype stage, of course, but they do exist, and may soon be used to produce nigh-uncrashable servers.

Hardware and architectural reliability may be important, but remember they were only about 20% of the pie as far as failures go. Admittedly, restarting nodes that have aging software could certainly reduce application and OS problems, [9] doing so doesn't really address the problems inherent in applications transmitting and receiving their data correctly. Nokia and Oasis have been busily researching additions to the SOAP protocol to build a much more reliable messaging framework. There are some critical reliability features that SOAP does not offer, no matter how reliable the hardware it communicates with is, the first of which being guaranteed delivery. [10, 11]

Guaranteed delivery partially embodies the communication concept of non-repudiation. If a message is sent to a destination there must be an assurance that it either will be received, or that an error will be reported to the sender. [10] A message could not then be sent and swallowed by the intermittent abyss that roams the Internet without the sender knowing. Duplicate elimination is another feature found to be necessary. Although uncomplicated duplicate elimination would be very useful, allowing multiple copies of the same information to be recognized and filtered out instead of being treated as new and unique. [10, 11] The next feature in the SOAP reliability hit parade is ordered exchange of information. [10, 11] Some applications may actually require that they receive information in the same order that it was sent from its source in order to provide a useful service. [10, 11] Crash tolerance and state synchronization are also two important messaging concerns. Crash tolerance assures that information provided by the communication protocol is always available regardless of physical machine failures, and state synchronization assures that if a message can not be sent or disappears in transit for any reason, both the source and the destination of the message make sure no partial state exists between them. [10, 11]

In early 2003, Nokia and Oasis both identified the need for a Message Exchange Protocol (MEP) that sits on top of SOAP and provides these qualities mentioned above. [10, 11] Only a year later, Oasis has generated a 77 page document describing a partial solution to provide the communication features they require. [12] In order to send a message with guaranteed delivery the Reliable Messaging Protocol (RMP) will first send out the message to its destination, but not without any type of regulation. When a message is handed to the RMP to be sent, it is tagged with an expiration date. The RMP then begins exhaustively sending and resending the message until either; it receives an acknowledgement from the receiver, a predefined number of retries have been looped through, or the message expires. [12] If any of these three conditions were met, the RMP sender will declare the message a fault, and if it receives confirmation from the RMP receiver that the message was indeed delivered, it will declare it successfully sent. [12] So, no matter what happens, the fate of a message can not remain unknown.

Screening out duplicate messages is also provided for. Each message has a message id associated with it. The sender will send messages with the same message id until it receives a response from the receiver, or until the message dies. [12] If a receiver ever receives a message whose id it has seen before, it can still take action on it, like sending an acknowledgment message, but it is not allowed to forward that identical message back up to the receiver's application layer. [12] This way, although duplicate messages can certainly be sent, and they are technically received, they are never actually

made available for processing by the user. The mechanisms provided by the guaranteed delivery and duplicate message interception frameworks make message ordering rather easy. If a message with a non-contiguous sequence number (which is incorporated into the message id) is received it is buffered until the contiguous message is received. [12] When that happens, all are forwarded in order to the application layer. [12] This entire reliable messaging architecture in fact closely resembles TCP/IP in many respects, albeit a more reliable and implemented in XML being transmitted via SOAP. Crash tolerance and state synchronization don't have solutions specifically declared in the Oasis reliable messaging document, however in one year three of the five reliability features required have been developed. Much like hardware, even though the messaging middleware's reliability may not be there in full force right now, it is on the near horizon. Middleware and messaging then isn't a particularly strong reliability concern. Unfortunately users aren't all that reliable, predictable or easy to examine or upgrade (and they bite when you try to), so it has often fallen to hardware, middleware and software to be able to handle all that users can throw at them. Since we have examined both hardware and middleware so far, let us descend deeper, into the realm of software.

Making any and all software and OSes in general totally reliable is about as easy as doing something… really, really, really hard… or maybe even harder. There are ways to insulate servers from software error, for instance the Backdoors project mentioned above, however software and OSes should be built with the concept of web services in mind so that providing those services later is not quite so much of a stunt. Microsoft is already heading in this direction. It has already apparently recognized that overloading a single application or OS with too many functions that it does not use simply creates more problems than it is worth. As an example, Thompson, Inc. is an international company supporting offices and functions in France, whose more profitable ventures include patent examination. In order to support the large amounts of data, simulations and reverse-engineering work done the Thompson servers need to be very reliable and work under a large amount of load for extended periods of time. [13] A systems and network administrator at the local Princeton facility of Thompson, Inc. said that the newer Windows Server editions are better at doing what they are made to do than the garden-variety Microsoft OSes. [13] He went on to say that, "… Windows XP is like the swiss army knife of operating systems, but if you're going to use the machine as a server, you'll never use most of it." [13] Gates himself at the Gartner Symposium/ITxpo 2004 had, "…touted Web services and XML as key to software development trends." [3] and said that "…over the next decade companies could reduce amount the amount of code they need to write by at least a factor of five." [3] Apparently Microsoft is moving full steam toward exploiting web services.

So, is it just Microsoft right now? Is reliability so bad that no other IT companies are jumping into the fray, or are putting very little money into it when they do? Well, if 'no other' means 'most' companies, and 'very little' means 'lots' of money, yes. Apparently web services offers so much promise and, for now, is so free of entrenched 'founding' companies that many companies are trying to jump in and be the first. Obviously Oasis and Nokia are both researching web services standards, but IT giant "Motorola has committed to the enterprisewide deployment of Web services," [14] and "No fewer than four organizations—Liberty Alliance, Oasis, W3C and WS-I—are vying to preside over the process [of building web services standards]…" [15] The web services

battle has seen unusual bedfellows, namely "…an uneasy alliance of IBM and Microsoft versus nearly everyone else." [15] In fact, IDC, the same research and benchmarking company whose analyses of IBM's servers was quoted earlier, "estimates that companies will do $2.2 billion worth of Web services projects in 2003 and $25 billion in 2008." [15] So, apparently all the heavyweights are racing like comets to be first in line, but we the people haven't seen any real results yet, is reliability the problem holding them back?

In a word; no. In two words; no way. In three words; not at all. In four words; this joke is tired already. But seriously, folks… the big players have more than enough money to throw into building servers that are reliable 24/7, besides from the analysis of current technology and technology on the way offered above, it is apparent that reliability will not be a big problem in the near future. The problem is, well, that the heavyweights have a lot to gain, or lose, depending on who is first out of the gate. Since the legalities of all this web service stuff, especially who will pay whom for what and how, is still very much up in the air, and no one vendor/standards agency/research group has come up with a full standard yet, web services won't be deployed in any large commercial role until the dust settles. In fact, Christopher Koch of CIO Magazine says it best: "Everyone wants Web services standards. CEOs think the technology will create new opportunities. CFOs believe it will save millions. Vendors see a pot of gold at the end of the Web services rainbow. And CIOs know that linking to customers and partners over the Internet will revolutionize both business and IT. So what's the holdup? The usual suspects: Politics. Ego. Suspicion. Fear. Greed." [15]

The organizations developing web services standards are themselves getting in each others' way and slowing down the development of web services by duplicating each others' work, patenting and litigating bits of web service standards and protocols already and generally being in the business of being businesses. IBM and Microsoft make a powerful yet odd team, and may very well come out on top of the heap. Until someone clambers out of the melee, and standing on the backs of their vanquished competitors, raises their litigiously bloodied fists into the air and bellows the clear, clarion, primal cry of corporate victory, web services will remain mostly unused curiosities… the unique province of academics, computer nerds (freelance academics), and the few daring front-runners.

References

[1] xSeries Group, IBM., The Necessity of Change Management in the Intel Server Space. ftp://ftp.pc.ibm.com/pub/pccbbs/pc_servers_pdf/the_necessity_of_change_management.pdf

[2] K. D. Larkowski.. The Top Ten Reasons CORBA is Boring. http://www.omg.org/corba-corner/02-Larkowski_Presentation.pdf

[3] D. Farber., Gates: In 2014, magic software, free hardware. *ZDNet Tech Update*. (29 March, 2004).

[4] IBM Corporation., IBM eserver xSeries 455. ftp://ftp.software.ibm.com/common/ssi/rep_sp/3/GM130443/GM130443.PDF (January, 2004)

[5] IBM Corporation, Systems Management for IBM eserver xSeries Servers 2003. http://www5.pc.ibm.com/us/me.nsf/US-webdocs/White+Paper%3A+Systems+Management+for+IBM+eServer+xSeries+Servers+--+2003# (May, 2003).

[6] T. P. Morgan., IDC Makes the Case for iSeries Wintel-Lintel Server Consolidation. *The Four Hundred: iSeries and AS/400 Insight.* OS/400 Edition, Volume 12, Number 30. (4 August, 2003).

[7] F. Sultan, A. Bohra, Y. Pan, et. al., Nonintrusive Failure Detection and Recovery for Internet Services using Backdoors. http://paul.rutgers.edu/~yufeipan/research/publications/bd.pdf

[8] Y. Pan., Demonstration of Backdoors Failure Detection and Recovery. *Rutgers University Computer Science Open House, Piscataway, NJ*. (March, 2004).

[9] K. S. Trivedi, K. Vaidyanathan., Symptom Based Fault Management in Software Systems: Analysis of Software Rejuvenation in Cluster Systems. *CACC Semi-Annual Research Review, Duke Univesity, Durham, NC.* (25 October, 2000) .

[10] Nokia., Web Service Reliability. http://xml.coverpages.org/NokiaWSReliability.pdf (March, 2003).

[11] Oasis., Web Service Reliability Requirements, Draft Version 0.02 http://lists.oasis-open.org/archives/wsrm/200304/doc00000.doc (2003).

[12] Oasis., Reliable Messaging: Recent Versions of WS-Reliability and WS-ReliableMessaing. *Cover Pages, Hosted by Oasis*. (March, 2004).

[13] T. Walsh., Personal Conversation. *New Brunswick, NJ*. (April, 2004).

[14] E. Varon., Calculated Risks. *CIO Magazine*, Oct 1., 2003 Issue. (October, 2003).

[15] C. Koch., The Battle for Web Services. *CIO Magazine*, Oct 1., 2003 Issue. (October, 2003).

[16] B. C. Ling, E. Kiciman, A. Fox., Session State: Beyond Soft State. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation.* San Francisco, CA. (March, 2004)