Fallacies of Distributed Computing Explained

(The more things change the more they stay the same)

Arnon Rotem-Gal-Oz

[This whitepaper is based on a series of blog posts that first appeared in Dr. Dobb's Portal www.ddj.com/dept/architect]

The software industry has been writing distributed systems for several decades. Two examples include The US Department of Defense ARPANET (which eventually evolved into the Internet) which was established back in 1969 and the SWIFT protocol (used for money transfers) was also established in the same time frame [Britton2001]. Nevertheless,

In 1994, Peter Deutsch, a sun fellow at the time, drafted 7 assumptions architects and designers of distributed systems are likely to make, which prove wrong in the long run - resulting in all sorts of troubles and pains for the solution and architects who made the assumptions. In 1997 James Gosling added another such fallacy [JDJ2004]. The assumptions are now collectively known as the "The 8 fallacies of distributed computing" [Gosling]:

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

This whitepaper will looks at each of these fallacies, explains them and checks their relevancy for distributed systems today.

The network is reliable

The first fallacy is "The network is reliable." Why is this a fallacy? Well, when was the last time you saw a switch fail? After all, even basic switches these days have MTBFs (Mean Time Between Failure) in the 50,000 operating hours and more.

For one, if your application is a mission critical 365x7 kind of application, you can just hit that failure--and Murphy will make sure it

happens in the most inappropriate moment. Nevertheless, most applications are not like that. So what's the problem?

Well, there are plenty of problems: Power failures, someone trips on the network cord, all of a sudden clients connect wirelessly, and so on. If hardware isn't enough--the software can fail as well, and it does.

The situation is more complicated if you collaborate with an external partner, such as an e-commerce application working with an external credit-card processing service. Their side of the connection is not under your direct control. Lastly there are security threats like DDOS attacks and the like.

What does that mean for your design?

On the infrastructure side, you need to think about hardware and software redundancy and weigh the risks of failure versus the required investment.

On the software side, you need to think about messages/calls getting lost whenever you send a message/make a call over the wire. For one you can use a communication medium that supplies full reliable messaging; WebsphereMQ or MSMQ, for example. If you can't use one, prepare to retry, acknowledge important messages, identify/ignore duplicates (or use idempotent messages), reorder messages (or not depend on message order), verify message integrity, and so on.

One note regarding WS-ReliableMessaging: The specification supports several levels of message guarantee--most once, at least once, exactly once and orders. You should remember though that it only takes care of delivering the message as long as the network nodes are up and running, it doesn’t handle persistency and you still need to take care of that (or use a vendor solution that does that for you) for a complete solution.

To sum up, the network is **Un**reliable and we as software architect/designers need to address that.

Latency is zero

The second fallacy of Distributed Computing is the assumption that "Latency is Zero". Latency is how much time it takes for data to move from one place to another (versus bandwidth which is how much data we can transfer during that time). Latency can be relatively good on a LAN--but latency deteriorates quickly when you move to WAN scenarios or internet scenarios.

Latency is more problematic than bandwidth. Here's a quote from a post by Ingo Rammer on latency vs. Bandwidth [Ingo] that illustrates this:

> "*But I think that it's really interesting to see that the end-to-end bandwidth increased by 1468 times within the last 11 years while the latency (the time a single ping takes) has only been improved tenfold. If this wouldn't be enough, there is even a natural cap on latency. The minimum round-trip time between two points of this earth is determined by the maximum speed of information transmission: the speed of light. At roughly 300,000 kilometers per second (3.6 * 10E12 teraangstrom per fortnight), it will always take at least 30 milliseconds to send a ping from Europe to the US and back, even if the processing would be done in real time.*"

You may think all is okay if you only deploy your application on LANs. However even when you work on a LAN with Gigabit Ethernet you should still bear in mind that the latency is much bigger then accessing local memory Assuming the latency is zero you can be easily tempted to assume making a call over the wire is almost like making a local calls--this is one of the problems with approaches like distributed objects, that provide "network transparency"--alluring you to make a lot of fine grained calls to objects which are actually remote and expensive (relatively) to call to.

Taking latency into consideration means you should strive to make as few as possible calls and assuming you have enough bandwidth (which will talk about next time) you'd want to move as much data out in each of this calls. There is a nice example illustrating the latency problem and what was done to solve it in Windows Explorer in http://blogs.msdn.com/oldnewthing/archive/2006/04/07/570801.aspx

Another example is AJAX. The AJAX approach allows for using the dead time the users spend digesting data to retrieve more data - however, you still need to consider latency. Let's say you are working on a new shiny AJAX front-end--everything looks just fine in your testing environment. It also shines in your staging environment passing the load tests with flying colors. The application can still fail miserably on the production environment if you fail to test for latency problems--retrieving data in the background is good but if you can't do that fast enough the application would still stagger and will be unresponsive.… (You can read more on AJAX and latency here.) [RichUI]

You can (and should) use tools like Shunra Virtual Enterprise, Opnet Modeler and many others to simulate network conditions and

understand system behavior thus avoiding failure in the production system.

Bandwidth is infinite

The next Distributed Computing Fallacy is "Bandwidth Is Infinite." This fallacy, in my opinion, is not as strong as the others. If there is one thing that is constantly getting better in relation to networks it is bandwidth.

However, there are two forces at work to keep this assumption a fallacy. One is that while the bandwidth grows, so does the amount of information we try to squeeze through it. VoIP, videos, and IPTV are some of the newer applications that take up bandwidth. Downloads, richer UIs, and reliance on verbose formats (XML) are also at work--especially if you are using T1 or lower lines. However, even when you think that this 10Gbit Ethernet would be more than enough, you may be hit with more than 3 Terabytes of new data per day (numbers from an actual project).

The other force at work to lower bandwidth is packet loss (along with frame size). This quote which underscores this point very well:

> *"In the local area network or campus environment, rtt and packet loss are both usually small enough that factors other than the above equation set your performance limit (e.g. raw available link bandwidths, packet forwarding speeds, host CPU limitations, etc.). In the WAN however, rtt and packet loss are often rather large and something that the end systems can not control. Thus their only hope for improved performance in the wide area is to use larger packet sizes.*
>
> *Let's take an example: New York to Los Angeles. Round Trip Time (rtt) is about 40 msec, and let's say packet loss is 0.1% (0.001). With an MTU of 1500 bytes (MSS of 1460), TCP throughput will have an upper bound of about 6.5 Mbps! And no, that is not a window size limitation, but rather one based on TCP's ability to detect and recover from congestion (loss). With 9000 byte frames, TCP throughput could reach about 40 Mbps.*
>
> *Or let's look at that example in terms of packet loss rates. Same round trip time, but let's say we want to achieve a throughput of 500 Mbps (half a "gigabit"). To do that with 9000 byte frames, we would need a packet loss rate of no more than 1x10^-5. With 1500 byte frames, the required packet loss rate is down to*

> *2.8x10^-7! While the jumbo frame is only 6 times larger, it allows us the same throughput in the face of 36 times more packet loss." [WareOnEarth]*

Acknowledging the bandwidth is not infinite has a balancing effect on the implications of the the "Latency Is Zero" fallacy; that is, if acting on the realization the latency is not zero we modeled few large messages. Bandwidth limitations direct us to strive to limit the size of the information we send over the wire.

The main implication then is to consider that in the production environment of our application there may be bandwidth problems which are beyond our control. And we should bear in mind how much data is expected to travel over the wise.

The recommendation I made in my previous post--to try to simulate the production environment--holds true here as well.

The Network is Secure

Peter Deutsch introduced the Distributed Computing Fallacies back in 1991. You'd think that in the 15 years since then that "the Network is secure" would no longer be a fallacy.

Unfortunately, that's not the case--and not because the network is now secure. No one would be naive enough to assume it is. Nevertheless, a few days ago I began writing a report about a middleware product some vendor tried to inflict on us that has no regard whatsoever to security! Well that is just anecdotal evidence, however.

Statistics published at Aladdin.com [Aladdin] shows that:

> ***"For 52% of the networks the perimeter is the only defense***
>
> *According to Preventsys and Qualys, 52% of chief information security officers acknowledged having a "Moat & Castle" approach to their overall network security . They admitted that once the perimeter security is penetrated, their networks are at risk. Yet, 48% consider themselves to be "proactive" when it comes to network security and feel that they have a good grasp on their enterprise's security posture. 24% felt their security was akin to Fort Knox (it would take a small army to get through), while 10% compared their network security to Swiss cheese (security holes inside and out). The remaining 14% of respondents described their current network security as being*

> *locked down on the inside, but not yet completely secured to the outside. Preventsys and Qualys also found that 46% of security officers spend more than a third of their day, and in some cases as much as 7 hours, analyzing reports generated from their various security point solutions. "*

In case you just landed from another planet **the network is far from being secured**. Here are few statistics to illustrate that:

> Through the continual 24x7 monitoring of hundreds of Fortune 1000 companies, RipTech has discovered several extremely relevant trends in information security. Among them:
>
> 1. General Internet attack trends are showing a 64% annual rate of growth
> 2. The average company experienced 32 attacks per week over the past 6 months
> 3. Attacks during weekdays increased in the past 6 months" [RipTech].

When I tried to find some updated incident statistics, I came up with the following [CERT]:

> Note: Given the widespread use of automated attack tools, attacks against Internet-connected systems have become so commonplace that counts of the number of incidents reported provide little information with regard to assessing the scope and impact of attacks. Therefore, as of 2004, we will no longer publish the number of incidents reported. Instead, we will be working with others in the community to develop and report on more meaningful metrics" (the number of incidents for 2003 was 137539 incidents...)

Lastly Aladdin claims that the costs of Malware for 2004 (Viruses, Worms, Trojans etc.) are estimated between $169 billion and $204 billion. [Aladdin]

The implications of network (in) security are obvious--you need to build security into your solutions from Day 1. I mentioned in a previous blog post that security is a system quality attribute that needs to be taken into consideration starting from the architectural level. There are dozens of books that talk about security and I cannot begin to delve into all the details in a short blog post.

In essence you need to perform threat modeling to evaluate the security risks. Then following further analyses decide which risk are

should be mitigated by what measures (a tradeoff between costs, risks and their probability). Security is usually a multi-layered solution that is handled on the network, infrastructure, and application levels.

As an architect you might not be a security expert--but you still need to be aware that security is needed and the implications it may have (for instance, you might not be able to use multicast, user accounts with limited privileges might not be able to access some networked resource etc.)

Topology doesn't change

The fifth Distributed Computing Fallacy is about network topology. "Topology doesn't change." That's right, it doesn't--as long as it stays in the test lab.

When you deploy an application in the wild (that is, to an organization), the network topology is usually out of your control. The operations team (IT) is likely to add and remove servers every once in a while and/or make other changes to the network ("this is the new Active Directory we will use for SSO ; we're replacing RIP with OSPF and this application's servers are moving into area 51" and so on). Lastly there are server and network faults which can cause routing changes.

When you're talking about clients, the situation is even worse. There are laptops coming and going, wireless ad-hoc networks , new mobile devices. In short, topology is changing **constantly**.

What does this mean for the applications we write? Simple. Try not to depend on specific endpoints or routes, if you can't be prepared to renegotiate endpoints. Another implication is that you would want to either provide location transparency (e.g. using an ESB, multicast) or provide discovery services (e.g. a Active Directory/JNDI/LDAP).

Another strategy is to abstract the physical structure of the network. The most obvious example for this is DNS names instead of IP addresses. Recently I moved my (other) blog from one hosting service to another. The transfer went without a hitch as I had both sites up an running. Then when the DNS routing tables were updated (it takes a day or two to the change to ripple) readers just came to the new site without knowing the routing (topology) changed under their feet.

An interesting example is moving from WS-Routing to WS-Addressing. In WS-Routing a message can describes it own routing path--this assumes that a message can know the path it needs to travel in advance. The topology doesn't change (this also causes a security

vulnerability--but that's another story) where the newer WS-Addressing relies on "Next Hop" routing (the way TCP/IP works) which is more robust.

Another example is routing in SQL Server Service Broker. The problematic part is that the routes need to be set inside service broker. This is problematic since IT now has to remember to go into Service Broker and update routing tables when topology changes. However, to mitigate this problem the routing relies on next-hop semantics and it allows for specifying the address by DNS name.

There is one administrator

The sixth Distributed Computing Fallacy is "There is one administrator". You may be able to get away with this fallacy if you install your software on small, isolated LANs (for instance, a single person IT "group" with no WAN/Internet). However, for most enterprise systems the reality is much different.

The IT group usually has different administrators, assigned according to expertise--databases, web servers, networks, Linux, Windows, Main Frame and the like. This is the easy situation. The problem is occurs when your company collaborates with external entities (for example, connecting with a business partner), or if your application is deployed for Internet consumption and hosted by some hosting service and the application consumes external services (think Mashups). In these situations, the other administrators are not even under your control and they may have their own agendas/rules.

At this point you may say "Okay, there is more than one administrator. But why should I care?" Well, as long as everything works, maybe you don't care. You do care, however, when things go astray and there is a need to pinpoint a problem (and solve it). For example, I recently had a problem with an ASP.NET application that required full trust on a hosting service that only allowed medium trust--the application had to be reworked (since changing host service was not an option) in order to work.

Furthermore, you need to understand that the administrators will most likely not be part of your development team so we need provide them with tools to diagnose and find problems. This is essential when the application involves more than one company ("Is it their problem or our's?"). A proactive approach is to also include tools for monitoring on-going operations as well; for instance, to allow administrators identify problems when they are small--before they become a system failure.

Another reason to think about multiple administrators is upgrades. How are you going to handle them? How are you going to make sure that the different parts of our application (distributed, remember?) are synchronized and can actually work together; for example, does the current DB schema match the current O/R mapping and object model? Again this problem aggravates when third parties are involved. Can your partner continue to interop with our system when we made changes to the public contract (in an SOA) so, for example, you need to think about backward compatibility (or maybe even forward compatibility) when designing interoperability contracts.

To sum up, when there is more than one administrator (unless we are talking about a simple system and even that can evolve later if it is successful), you need to remember that administrators can constrain your options (administrators that sets disk quotas, limited privileges, limited ports and protocols and so on), and that you need to help them manage your applications.

Transport cost is zero

On to Distributed Computing Fallacy number 7--"Transport cost is zero". There are a couple of ways you can interpret this statement, both of which are false assumptions.

One way is that going from the application level to the transport level is free. This is a fallacy since we have to do marshaling (serialize information into bits) to get data unto the wire, which takes both computer resources and adds to the latency. Interpreting the statement this way emphasizes the "Latency is Zero" fallacy by reminding us that there are additional costs (both in time and resources).

The second way to interpret the statement is that the costs (as in cash money) for setting and running the network are free. This is also far from being true. There are costs--costs for buying the routers, costs for securing the network, costs for leasing the bandwidth for internet connections, and costs for operating and maintaining the network running. Someone, somewhere will have to pick the tab and pay these costs.

Imagine you have successfully built Dilbert's Google-killer search engine [Adams] (maybe using latest Web 2.0 bells-and-whistles on the UI) but you will fail if you neglect to take into account the costs that are needed to keep your service up, running, and responsive (E3 Lines, datacenters with switches, SANs etc.). The takeaway is that even in situations you think the other fallacies are not relevant to your situation because you rely on existing solutions ("yeah, we'll just

deploy Cisco's HSRP protocol and get rid of the network reliability problem") you may still be bounded by the costs of the solution and you'd need to solve your problems using more cost-effective solutions.

The network is homogeneous.

The eighth and final Distributed Computing fallacy is "The network is homogeneous."

While the first seven fallacies were coined by Peter Deutsch, I read [JDJ2004] that the eighth fallacy was added by James Gosling six years later (in 1997).

Most architects today are not naïve enough to assume this fallacy. Any network, except maybe the very trivial ones, are not homogeneous. Heck, even my home network has a Linux based HTPC, a couple of Windows based PCs, a (small) NAS, and a WindowMobile 2005 device--all connected by a wireless network. What's true on a home network is almost a certainty in enterprise networks. I believe that a homogeneous network today is the exception, not the rule. Even if you managed to maintain your internal network homogeneous, you will hit this problem when you would try to cooperate with a partner or a supplier.

Assuming this fallacy should not cause too much trouble at the lower network level as IP is pretty much ubiquitous (e.g. even a specialized bus like Infiniband has an IP-Over-IB implementation, although it may result in suboptimal use of the non-native IP resources.

It is worthwhile to pay attention to the fact the network is not homogeneous at the application level. The implication of this is that you have to assume interoperability will be needed sooner or later and be ready to support it from day one (or at least design where you'd add it later).

Do not rely on proprietary protocols--it would be harder to integrate them later. Do use standard technologies that are widely accepted; the most notable examples being XML or Web Services. By the way, much of the popularity of XML and Web Services can be attributed to the fact that both these technologies help alleviate the affects of the heterogeneity of the enterprise environment.

To sum up, most architects/designers today are aware of this fallacy, which is why interoperable technologies are popular. Still it is something you need to keep in mind especially if you are in a situation that mandates use of proprietary protocols or transports.

Summary

With almost 15 years since the fallacies were drafted and more than 40 years since we started building distributed systems – the characteristics and underlying problems of distributed systems remain pretty much the same. What is more alarming is that architects, designers and developers are still tempted to wave some of these problems off thinking technology solves everything.

Remember that (successful) applications evolve and grow so even if things look Ok for a while if you don't pay attention to the issues covered by the fallacies they will rear their ugly head and bite you.

I hope that reading this paper both helped explain what the fallacies mean as well as provide some guidance on what to do to avoid their implications.

References

[Britton2001] IT Architecture & Middleware, C. Britton, Addison-Wesley 2001, ISBN 0-201-70907-4

[JDJ2004]. http://java.sys-con.com/read/38665.htm

[Gosling] http://blogs.sun.com/roller/page/jag

[Ingo] http://blogs.thinktecture.com/ingo/archive/2005/11/08/LatencyVsBandwidth.aspx

[RichUI] http://richui.blogspot.com/2005/09/ajax-latency-problems-myth-or-reality.html

[WareOnEarth] http://sd.wareonearth.com/~phil/jumbo.html

[Aladdin] http://www.esafe.com/home/csrt/statistics/statistics_2005.asp

[RipTech] http://www.riptech.com/

[CERT] http://www.cert.org/stats/#incidents\

[Adams] http://www.dilbert.com/comics/dilbert/archive/dilbert-20060516.html