Kekeliruan dari Computing Distributed Dijelaskan

(Semakin banyak hal berubah semakin mereka tetap sama)

Arnon Rotem-Gal-Oz

[Whitepaper ini didasarkan pada serangkaian posting blog yang pertama kali muncul di Portal Dr. Dobb www.ddj.com/dept/architect]

Industri perangkat lunak telah tertulis sistem terdistribusi selama beberapa dekade. Dua contoh termasuk The US Department of Defense ARPANET (yang akhirnya berkembang menjadi internet) yang didirikan kembali pada tahun 1969 dan SWIFT protokol (digunakan untuk transfer uang) juga didirikan di rentang waktu yang sama [Britton2001]. Namun,

Pada tahun 1994, Peter Deutsch, sesama matahari pada saat itu, menyusun 7 asumsi arsitek dan desainer sistem terdistribusi cenderung membuat, yang terbukti salah dalam jangka panjang - yang mengakibatkan segala macam masalah dan nyeri untuk solusi dan arsitek yang membuat asumsi. Pada tahun 1997 James Gosling menambahkan kekeliruan seperti yang lain [JDJ2004]. Asumsi sekarang kolektif dikenal sebagai "The 8 kesalahan-kesalahan dari komputasi terdistribusi" [Gosling]:

1. jaringan yang handal.
2. Latency adalah nol.
3. Bandwidth tak terbatas.
4. jaringan aman.
5. Topologi tidak berubah.
6. Ada satu administrator.
7. biaya transportasi adalah nol.
8. jaringan homogen.

akan whitepaper ini terlihat pada setiap kesalahan-kesalahan ini, menjelaskan mereka dan memeriksa relevansi mereka untuk sistem terdistribusi hari ini.

jaringan yang handal

Kesalahan pertama adalah "Jaringan ini dapat diandalkan." Mengapa ini kekeliruan? Nah, kapan terakhir kali Anda melihat sebuah saklar gagal? Setelah semua, bahkan switch dasar hari ini memiliki MTBFs (Mean Time Between Failure) di
50.000 jam operasi dan banyak lagi.

Untuk satu, jika aplikasi Anda adalah misi 365x7 kritis jenis aplikasi, Anda dapat tekan saja bahwa kegagalan - dan Murphy akan memastikan itu

terjadi di saat yang paling tidak pantas. Namun demikian, sebagian besar aplikasi tidak seperti itu. Jadi apa masalahnya?

Nah, ada banyak masalah: kegagalan Power, seseorang perjalanan pada kabel jaringan, semua dari klien tiba-tiba terhubung secara nirkabel, dan sebagainya. Jika hardware tidak cukup - perangkat lunak dapat gagal juga, dan itu tidak.

Situasi lebih rumit jika Anda berkolaborasi dengan mitra eksternal, seperti sebuah aplikasi e-commerce bekerja dengan layanan pemrosesan kartu kredit eksternal. sisi mereka sambungan tidak di bawah kendali langsung Anda. Terakhir ada ancaman keamanan seperti serangan DDOS dan sejenisnya.

Apa artinya untuk desain Anda?

Di sisi infrastruktur, Anda harus berpikir tentang perangkat keras dan perangkat lunak redundansi dan mempertimbangkan risiko kegagalan versus investasi yang diperlukan.

Di sisi perangkat lunak, Anda harus berpikir tentang pesan / panggilan tersesat setiap kali Anda mengirim pesan / membuat panggilan atas kawat. Untuk satu Anda dapat menggunakan media komunikasi yang memasok pesan handal penuh; WebsphereMQ atau MSMQ, misalnya. Jika Anda tidak dapat menggunakan satu, mempersiapkan diri untuk mencoba lagi, mengakui pesan penting, mengidentifikasi / mengabaikan duplikat (atau menggunakan idempoten pesan), menyusun ulang pesan (atau tidak tergantung pada urutan pesan), memverifikasi integritas pesan, dan sebagainya.

Satu catatan tentang WS-ReliableMessaging : Spesifikasi mendukung beberapa tingkat jaminan pesan - paling banyak sekali, setidaknya sekali, tepat sekali dan perintah. Anda harus ingat bahwa meskipun hanya mengurus menyampaikan pesan selama node jaringan yang berdiri dan berjalan, tidak menangani persistensi dan Anda masih perlu mengurus itu (atau menggunakan solusi vendor yang melakukan itu untuk Anda) untuk solusi lengkap.

Singkatnya, jaringan un handal dan kita sebagai software arsitek / desainer perlu untuk mengatasi itu.

Latency adalah nol

Kesalahan kedua Computing Distributed adalah asumsi bahwa "Latency adalah nol". Latency adalah berapa banyak waktu yang diperlukan untuk data untuk berpindah dari satu tempat ke tempat lain (versus bandwith yang adalah berapa banyak data yang kita dapat mentransfer selama waktu itu). Latency dapat relatif baik pada LAN - tapi latency memburuk dengan cepat ketika Anda pindah ke WAN skenario atau skenario internet.

Latency lebih bermasalah daripada bandwidth. Berikut kutipan dari posting oleh Ingo Rammer pada latency vs Bandwidth [Ingo] yang menggambarkan ini:

> " *B ut Saya berpikir bahwa itu benar-benar menarik untuk melihat bahwa bandwidth end-to-end meningkat 1468 kali dalam 11 tahun terakhir sementara latency (waktu ping tunggal memakan waktu) hanya telah ditingkatkan sepuluh kali lipat. Jika w ini ouldn't cukup, bahkan ada topi alami pada latency. Waktu pulang-pergi minimum antara dua titik dari bumi ini ditentukan oleh kecepatan maksimum transmisi informasi: kecepatan cahaya. Pada kira-kira 300.000 kilometer per detik (3,6 * 10E12 teraangstrom per dua minggu), itu akan selalu mengambil setidaknya 30 milidetik untuk mengirim ping dari Eropa ke AS dan kembali, bahkan jika pengolahan akan dilakukan secara real time."*

Anda mungkin berpikir semua baik-baik saja jika Anda hanya menyebarkan aplikasi Anda pada LAN. Namun bahkan ketika Anda bekerja pada LAN dengan Gigabit Ethernet Anda harus tetap diingat bahwa latency jauh lebih besar kemudian mengakses memori lokal asumsi latency adalah nol Anda dapat dengan mudah tergoda untuk menganggap membuat panggilan atas kawat hampir seperti membuat panggilan lokal - ini adalah salah satu masalah dengan pendekatan seperti objek terdistribusi, yang menyediakan "transparansi jaringan" - memikat Anda untuk membuat banyak panggilan berbutir halus untuk benda-benda yang sebenarnya jauh dan mahal (relatif) untuk panggilan ke.

Mengambil latency menjadi pertimbangan berarti Anda harus berusaha untuk membuat sesedikit panggilan mungkin dan dengan asumsi Anda memiliki cukup bandwidth (yang akan berbicara tentang waktu berikutnya) Anda ingin memindahkan banyak data di masing-masing ini panggilan. Ada contoh bagus yang menggambarkan masalah latency dan apa yang telah dilakukan untuk menyelesaikannya dalam Windows Explorer di

http://blogs.msdn.com/oldnewthing/archive/2006/04/07/570801.aspx

Contoh lain adalah AJAX. AJAX Pendekatan memungkinkan untuk menggunakan waktu mati pengguna menghabiskan mencerna data untuk mengambil lebih banyak data - namun, Anda masih perlu mempertimbangkan latency. Katakanlah Anda bekerja pada baru mengkilap AJAX front-end - semuanya terlihat baik-baik saja di lingkungan pengujian Anda. Hal ini juga bersinar di lingkungan pementasan Anda melewati tes beban dengan terbang warna. Aplikasi ini masih bisa gagal total pada lingkungan produksi jika Anda gagal untuk menguji data retrieving problems-- latency di latar belakang baik tetapi jika Anda tidak dapat melakukan itu cukup cepat aplikasi masih akan sempoyongan dan akan tidak responsif. ... (Anda dapat membaca lebih lanjut tentang AJAX dan latency sini .) [RichUI]

Anda dapat (dan harus) menggunakan alat-alat seperti Shunra Virtual Enterprise . Opnet Modeler dan banyak orang lain untuk mensimulasikan kondisi jaringan dan

memahami perilaku sistem sehingga menghindari kegagalan dalam sistem produksi.

Bandwidth tak terbatas

Berikutnya Distributed Computing Kekeliruan adalah "Bandwidth Apakah Tak Terbatas." kekeliruan ini, menurut pendapat saya, tidak sekuat seperti yang lain. Jika ada satu hal yang terus semakin baik dalam kaitannya dengan jaringan itu adalah bandwidth.

Namun, ada dua kekuatan yang bekerja untuk menjaga asumsi ini kekeliruan. Salah satunya adalah bahwa sementara bandwidth tumbuh, begitu pula jumlah informasi kami mencoba untuk memeras melalui itu. VoIP, video, dan IPTV adalah beberapa aplikasi baru yang mengambil bandwidth. Download, UIS kaya, dan ketergantungan pada format verbose (XML) juga bekerja-terutama jika Anda menggunakan T1 atau garis bawah. Namun, bahkan ketika Anda berpikir bahwa Ethernet 10Gbit ini akan lebih dari cukup, Anda mungkin akan memukul dengan lebih dari 3 terabyte data baru per hari (angka dari proyek yang sebenarnya).

Kekuatan lain di tempat kerja untuk bandwidth yang lebih rendah adalah packet loss (bersama dengan ukuran frame). Kutipan ini yang menggarisbawahi titik ini sangat baik:

> *"Dalam jaringan area atau kampus lingkungan setempat, rtt dan packet loss yang baik biasanya cukup kecil bahwa faktor-faktor lain selain persamaan di atas menetapkan batas Anda kinerja (misalnya baku bandwidth link yang tersedia, kecepatan forwarding paket, keterbatasan CPU tuan rumah, dll). Dalam WAN namun, rtt dan packet loss sering agak besar dan sesuatu yang sistem akhir tidak dapat mengendalikan. Jadi satu-satunya harapan mereka untuk meningkatkan kinerja di wilayah yang luas adalah dengan menggunakan ukuran paket yang lebih besar.*

> *Mari kita ambil contoh: New York ke Los Angeles. Bulat Trip Time (RTT) adalah sekitar 40 msec, dan katakanlah packet loss adalah 0,1% (0,001). Dengan MTU 1500 byte (MSS dari 1460), TCP throughput yang akan memiliki batas atas sekitar 6,5 Mbps! Dan tidak, itu bukan batasan ukuran jendela, melainkan satu berdasarkan kemampuan TCP untuk mendeteksi dan pulih dari kemacetan (rugi). Dengan 9000 frame byte, TCP throughput yang bisa mencapai sekitar 40 Mbps.*

> *Atau mari kita lihat contoh yang dalam hal tingkat packet loss. Sama putaran waktu perjalanan, tetapi katakanlah kita ingin mencapai throughput 500 Mbps (setengah "gigabit"). Untuk melakukan itu dengan 9000 frame byte, kita akan membutuhkan tingkat packet loss tidak lebih dari 1x10 ^ -5. Dengan 1500 frame byte, tingkat kerugian paket yang dibutuhkan adalah ke*

2.8x10 ^ -7! Sedangkan frame jumbo hanya 6 kali lebih besar, memungkinkan kita throughput yang sama dalam menghadapi 36 kali lebih packet loss."[WareOnEarth]

Mengakui bandwidth tidak terbatas memiliki efek menyeimbangkan pada implikasi dari para "Latency Is Nol" fallacy; yaitu, jika bertindak atas realisasi latency tidak nol kita dimodelkan beberapa pesan yang besar. keterbatasan bandwidth mengarahkan kita untuk berusaha untuk membatasi ukuran informasi yang kami kirim melalui kawat.

Implikasi utama kemudian adalah untuk mempertimbangkan bahwa dalam lingkungan produksi aplikasi kita mungkin ada masalah bandwith yang berada di luar kendali kita. Dan kita harus ingat berapa banyak data diharapkan untuk melakukan perjalanan selama bijaksana.

Rekomendasi saya membuat di posting saya sebelumnya - untuk mencoba untuk mensimulasikan lingkungan produksi - berlaku di sini juga.

Jaringan adalah Aman

Peter Deutsch memperkenalkan Computing Kekeliruan Distributed kembali
1991. Anda akan berpikir bahwa dalam 15 tahun sejak itu bahwa "Jaringan adalah aman" tidak akan lagi menjadi suatu kesalahan.

Sayangnya, itu tidak terjadi - dan bukan karena jaringan sekarang aman. Tidak ada yang akan cukup naif untuk menganggap itu adalah. Namun demikian, beberapa hari yang lalu saya mulai menulis laporan tentang produk middleware beberapa vendor yang mencoba untuk menimbulkan pada kita yang tidak memiliki kaitan apapun untuk keamanan! Nah itu hanya bukti anekdot, namun.

Statistik diterbitkan di Aladdin.com [Aladdin] menunjukkan bahwa:

"Untuk 52% dari jaringan perimeter adalah satu-satunya pertahanan

Menurut Preventsys dan Qualys, 52% dari petugas keamanan informasi kepala mengakui memiliki "Moat & Castle" pendekatan keamanan jaringan mereka secara keseluruhan. Mereka mengakui bahwa setelah keamanan perimeter ditembus, jaringan mereka beresiko. Namun, 48% menganggap diri mereka "proaktif" ketika datang ke keamanan jaringan dan merasa bahwa mereka memiliki pemahaman yang baik tentang postur keamanan perusahaan mereka. 24% merasa keamanan mereka itu mirip dengan Fort Knox (itu akan mengambil pasukan kecil untuk melewati), sementara 10% dibandingkan keamanan jaringan mereka ke keju Swiss (lubang keamanan di dalam dan luar). Sisanya 14% responden menggambarkan keamanan jaringan mereka saat ini sebagai

> *dikurung di dalam, tetapi belum sepenuhnya diamankan ke luar. Preventsys dan Qualys juga menemukan bahwa 46% dari petugas keamanan menghabiskan lebih dari sepertiga hari mereka, dan dalam beberapa kasus sebanyak 7 jam, laporan menganalisis dihasilkan dari berbagai solusi titik keamanan mereka. "*

Dalam kasus Anda baru saja mendarat dari planet lain **jaringan masih jauh dari aman.** Berikut adalah beberapa statistik untuk menggambarkan bahwa:

> Melalui pemantauan 24x7 terus-menerus ratusan Fortune 1000 perusahaan, RipTech telah menemukan beberapa kecenderungan sangat relevan dalam keamanan informasi. Diantara mereka:
>
> 1. Tren Serangan Internet umum menunjukkan tingkat tahunan 64% dari pertumbuhan
> 2. Rata-rata perusahaan mengalami 32 serangan per minggu selama masa lalu 6 bulan
> 3. Serangan selama hari kerja meningkat dalam 6 bulan terakhir"
>
> [RipTech].

Ketika saya mencoba untuk menemukan beberapa statistik insiden diperbarui, saya datang dengan [berikut CERT]:

> Catatan: Mengingat meluasnya penggunaan alat serangan otomatis, serangan terhadap sistem yang tersambung ke Internet telah menjadi begitu biasa yang penting dari jumlah insiden dilaporkan memberikan sedikit informasi berkaitan dengan menilai lingkup dan dampak serangan. Oleh karena itu, pada tahun 2004, kami tidak akan lagi menerbitkan jumlah insiden yang dilaporkan. Sebaliknya, kita akan bekerja dengan orang lain dalam masyarakat untuk mengembangkan dan melaporkan metrik lebih berarti"(jumlah insiden untuk tahun 2003 adalah 137.539 insiden ...)

akhirnya Aladdin mengklaim bahwa biaya Malware untuk tahun 2004 (Virus, Worms, Trojan dll) diperkirakan antara $ 169 miliar dan $ 204 miliar. [Aladdin]

Implikasi dari jaringan (di) keamanan yang jelas - Anda perlu membangun keamanan ke dalam solusi Anda dari hari 1. saya sebutkan di posting blog sebelumnya bahwa keamanan adalah atribut kualitas sistem yang perlu dipertimbangkan mulai dari tingkat arsitektur . Ada puluhan buku yang berbicara tentang keamanan dan aku tidak bisa mulai untuk menyelidiki semua rincian dalam posting blog singkat.

Pada intinya Anda perlu melakukan pemodelan ancaman untuk mengevaluasi risiko keamanan. Kemudian berikut analisis lebih lanjut memutuskan mana risiko yang

harus diatasi dengan langkah-langkah apa (tradeoff antara biaya, risiko dan probabilitas mereka). Keamanan biasanya solusi multi-layered yang ditangani pada tingkat jaringan, infrastruktur, dan aplikasi.

Sebagai seorang arsitek Anda mungkin tidak menjadi ahli keamanan - tetapi Anda masih perlu menyadari bahwa keamanan diperlukan dan implikasi mungkin memiliki (misalnya, Anda mungkin tidak dapat menggunakan multicast, account pengguna dengan hak istimewa terbatas mungkin tidak dapat mengakses beberapa jaringan sumber daya dll)

Topologi tidak berubah

Kelima Distributed Computing Kekeliruan adalah tentang topologi jaringan. "Topologi tidak berubah." Itu benar, itu doesnt - asalkan tetap di lab uji.

Ketika Anda menyebarkan aplikasi di alam liar (yaitu, untuk sebuah organisasi), topologi jaringan biasanya di luar kendali Anda. Tim operasi (TI) cenderung menambah dan menghapus server setiap sekali dalam beberapa saat dan / atau membuat perubahan lain ke jaringan ("ini adalah Active Directory yang baru kita akan gunakan untuk SSO; kami mengganti RIP dengan OSPF dan ini server aplikasi yang pindah ke daerah 51" dan seterusnya). Terakhir ada server dan jaringan kesalahan yang dapat menyebabkan perubahan routing.

Ketika Anda sedang berbicara tentang klien, situasinya bahkan lebih buruk. Ada laptop datang dan pergi, jaringan ad-hoc nirkabel, perangkat mobile baru. Singkatnya, topologi berubah **selalu.**

Apa artinya ini bagi aplikasi kita menulis? Sederhana. Cobalah untuk tidak bergantung pada endpoint atau rute tertentu, jika Anda tidak dapat dipersiapkan untuk menegosiasikan kembali endpoint. Implikasi lain adalah bahwa Anda akan ingin baik memberikan transparansi lokasi (misalnya menggunakan ESB, multicast) atau menyediakan jasa penemuan (misalnya Active Directory / JNDI / LDAP).

Strategi lain adalah untuk abstrak struktur fisik jaringan. Contoh yang paling jelas untuk ini adalah nama-nama DNS bukan alamat IP. Baru saja saya pindah saya (lainnya) blog dari satu layanan hosting yang lain. transfer berjalan lancar karena saya harus kedua situs sebuah berjalan. Kemudian ketika DNS tabel routing diperbarui (dibutuhkan satu atau dua hari untuk perubahan untuk riak) pembaca hanya datang ke situs baru tanpa mengetahui routing (topologi) berubah di bawah kaki mereka.

Contoh yang menarik bergerak dari WS-Routing ke WS-Addressing . Dalam WS-Routing pesan dapat menjelaskan itu sendiri Routing jalan - ini mengasumsikan bahwa pesan dapat mengetahui jalur yang dibutuhkan untuk melakukan perjalanan di muka. Topologi tidak berubah (ini juga menyebabkan keamanan

kerentanan - tapi itu cerita lain) di mana WSAddressing baru bergantung pada "Berikutnya Hop" routing (cara TCP / IP karya) yang lebih kuat.

Contoh lain adalah routing di SQL Server Service Broker. Bagian bermasalah adalah bahwa rute perlu ditetapkan dalam layanan broker. Hal ini bermasalah karena IT sekarang harus ingat untuk masuk ke Layanan Broker dan memperbarui tabel routing ketika perubahan topologi. Namun, untuk mengurangi masalah ini routing bergantung pada semantik hop berikutnya dan memungkinkan untuk menentukan alamat dengan nama DNS.

Ada satu administrator

Computing Kekeliruan keenam Distributed adalah "Ada satu administrator". Anda mungkin dapat lolos dengan kesalahan ini jika Anda menginstal perangkat lunak Anda pada kecil, LAN terisolasi (misalnya, satu orang IT "kelompok" tanpa WAN / Internet). Namun, untuk sebagian besar sistem perusahaan kenyataannya jauh berbeda.

Kelompok IT biasanya memiliki administrator yang berbeda, ditugaskan menurut keahlian - database, server web, jaringan, Linux, Windows, Frame Utama dan sejenisnya. Ini adalah situasi yang mudah. Masalahnya adalah terjadi ketika perusahaan Anda bekerja sama dengan entitas eksternal (misalnya, menghubungkan dengan mitra bisnis), atau jika aplikasi Anda dikerahkan untuk konsumsi internet dan diselenggarakan oleh beberapa layanan hosting dan aplikasi mengkonsumsi layanan eksternal (berpikir mashup). Dalam situasi ini, administrator lain bahkan tidak di bawah kendali Anda dan mereka mungkin memiliki agenda mereka sendiri / aturan.

Pada titik ini Anda mungkin berkata "Oke, ada lebih dari satu administrator. Tapi mengapa saya harus peduli?" Nah, selama semuanya bekerja, mungkin Anda tidak peduli. Anda melakukan perawatan, namun, ketika hal-hal sesat dan ada kebutuhan untuk menentukan masalah (dan memecahkannya). Sebagai contoh, saya baru-baru ini punya masalah dengan aplikasi ASP.NET yang diperlukan kepercayaan penuh pada layanan hosting yang hanya diperbolehkan trust- menengah

- aplikasi harus dikerjakan ulang (karena mengubah layanan host bukanlah pilihan) untuk bekerja.

Selain itu, Anda perlu memahami bahwa administrator akan kemungkinan besar tidak menjadi bagian dari tim pengembangan Anda sehingga kita perlu menyediakan mereka dengan alat untuk mendiagnosa dan menemukan masalah. Hal ini penting saat aplikasi melibatkan lebih dari satu perusahaan ("Apakah itu masalah mereka atau untuk kita?"). Sebuah pendekatan proaktif untuk juga menyertakan alat untuk monitoring on-akan operasi juga; misalnya, untuk memungkinkan administrator mengidentifikasi masalah ketika mereka kecil - sebelum mereka menjadi kegagalan sistem.

Alasan lain untuk berpikir tentang beberapa administrator adalah upgrade. Bagaimana Anda akan menangani mereka? Bagaimana Anda akan memastikan bahwa bagian-bagian yang berbeda dari aplikasi kita akan disinkronisasi dan benar-benar dapat bekerja sama (didistribusikan, ingat?); misalnya, apakah skema DB saat ini sesuai O / R pemetaan dan objek saat model? Sekali lagi masalah ini memperburuk ketika pihak ketiga yang terlibat. Dapat pasangan Anda terus Interop dengan sistem kami ketika kita membuat perubahan pada kontrak publik (dalam SOA) sehingga, misalnya, Anda harus berpikir tentang kompatibilitas (atau kompatibilitas bahkan mungkin ke depan) ketika merancang kontrak interoperabilitas.

Singkatnya, ketika ada lebih dari satu administrator (kecuali kita berbicara tentang sistem yang sederhana dan bahkan dapat berkembang kemudian jika berhasil), Anda perlu ingat bahwa administrator dapat membatasi pilihan Anda (administrator yang menetapkan kuota disk, terbatas hak istimewa, pelabuhan terbatas dan protokol dan sebagainya), dan bahwa Anda perlu untuk membantu mereka mengelola aplikasi Anda.

biaya transportasi adalah nol

Untuk Distributed Computing Kekeliruan nomor 7 - "Biaya Transportasi adalah nol". Ada beberapa cara Anda dapat menafsirkan pernyataan ini, yang keduanya asumsi yang salah.

Salah satu cara adalah bahwa pergi dari tingkat aplikasi ke tingkat transportasi gratis. Ini adalah kekeliruan karena kita harus melakukan marshaling (cerita bersambung informasi ke dalam bit) untuk mendapatkan data kepada kawat, yang mengambil baik sumber daya komputer dan menambah latency. Menafsirkan pernyataan cara ini menekankan "Latency adalah nol" kesalahan dengan mengingatkan kita bahwa ada biaya tambahan (baik dalam waktu dan sumber daya).

Cara kedua untuk menafsirkan pernyataan itu adalah bahwa biaya (seperti dalam uang tunai) untuk menetapkan dan menjalankan jaringan bebas. Hal ini juga jauh dari benar. Ada biaya - biaya untuk membeli router, biaya untuk mengamankan jaringan, biaya untuk sewa bandwidth untuk koneksi internet, dan biaya untuk operasi dan pemeliharaan jaringan berjalan. Seseorang, di suatu tempat akan harus memilih tab dan membayar biaya-biaya tersebut.

Bayangkan Anda telah berhasil membangun mesin pencari Google-pembunuh Dilbert ini [Adams] (mungkin menggunakan Web 2.0 terbaru lonceng-dan-peluit di UI) tetapi Anda akan gagal jika Anda lalai untuk memperhitungkan biaya yang diperlukan untuk menjaga layanan Anda up, berjalan, dan responsif (E3 Lines, pusat data dengan switch, SAN dll). takeaway adalah bahwa bahkan dalam situasi Anda berpikir kekeliruan lain yang tidak relevan dengan situasi Anda karena Anda mengandalkan solusi yang ada ("ya, kita hanya akan

menyebarkan protokol HSRP Cisco dan menyingkirkan jaringan masalah keandalan ") Anda mungkin masih dibatasi oleh biaya dari solusi dan Anda akan perlu untuk memecahkan masalah Anda menggunakan solusi yang lebih hemat biaya.

jaringan homogen.

Kedelapan dan terakhir Distributed Computing kesalahan adalah "Jaringan ini homogen."

Sementara tujuh pertama kekeliruan yang diciptakan oleh Peter Deutsch, saya Baca baca [JDJ2004] bahwa kesalahan kedelapan ditambahkan oleh James Gosling enam tahun kemudian (tahun 1997).

Kebanyakan arsitek saat ini tidak cukup naif untuk menganggap kesalahan ini. Jaringan, kecuali mungkin yang sangat sepele, tidak homogen. Heck, bahkan jaringan rumah saya memiliki HTPC berbasis Linux, beberapa PC berbasis Windows, (kecil) NAS, dan perangkat-WindowMobile 2005

- semua terhubung oleh jaringan wireless. Apa yang benar di jaringan rumah hampir kepastian dalam jaringan perusahaan. Saya percaya bahwa jaringan homogen hari ini adalah pengecualian, bukan aturan. Bahkan jika Anda berhasil mempertahankan jaringan internal Anda homogen, Anda akan terkena masalah ini ketika Anda akan mencoba untuk bekerja sama dengan mitra atau pemasok.

Dengan asumsi kesalahan ini seharusnya tidak menyebabkan terlalu banyak masalah di tingkat jaringan yang lebih rendah sebagai IP cukup banyak di mana-mana (misalnya bahkan bus khusus seperti Infiniband memiliki implementasi IP-Over-IB, meskipun dapat mengakibatkan penggunaan suboptimal dari sumber daya non-pribumi IP.

Hal ini bermanfaat untuk memperhatikan fakta jaringan tidak homogen di tingkat aplikasi. Implikasi dari ini adalah bahwa Anda harus menganggap interoperabilitas akan dibutuhkan cepat atau lambat dan siap untuk mendukungnya dari satu hari (atau setidaknya desain di mana Anda akan menambahkan nanti).

Jangan mengandalkan protokol proprietary - akan sulit untuk mengintegrasikan mereka nanti. Jangan menggunakan teknologi standar yang diterima secara luas; contoh yang paling menonjol menjadi XML atau Web Services. By the way, banyak popularitas XML dan Web Services dapat dikaitkan dengan fakta bahwa kedua teknologi ini membantu meringankan mempengaruhi heterogenitas lingkungan perusahaan.

Singkatnya, kebanyakan arsitek / desainer hari ini menyadari kekeliruan ini, yang mengapa teknologi interoperable yang populer. Masih itu adalah sesuatu yang Anda butuhkan untuk diingat terutama jika Anda berada dalam situasi yang mengamanatkan penggunaan protokol proprietary atau transportasi.

Ringkasan

Dengan hampir 15 tahun sejak kekeliruan ditulis dan lebih dari 40 tahun sejak kami mulai membangun sistem terdistribusi - karakteristik dan masalah yang mendasari sistem terdistribusi tetap hampir sama. Apa yang lebih mengkhawatirkan adalah bahwa arsitek, desainer dan pengembang masih tergoda untuk gelombang beberapa masalah ini off teknologi berpikir memecahkan segalanya.

Ingat bahwa (sukses) aplikasi berkembang dan tumbuh bahkan jika hal mencari Ok untuk sementara waktu jika Anda tidak memperhatikan isu-isu yang dicakup oleh kesalahan-kesalahan mereka akan belakang kepala jelek mereka dan menggigit Anda.

Saya berharap bahwa membaca tulisan ini baik membantu menjelaskan apa kesalahan-kesalahan berarti serta memberikan beberapa petunjuk tentang apa yang harus dilakukan untuk menghindari implikasi mereka.

Referensi

[Britton2001] IT Architecture & Middleware, C. Britton, AddisonWesley 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/LatencyVsBan dwidth.aspx

[RichUI] http://richui.blogspot.com/2005/09/ajax-latency-problemsmyth-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