AMERICAN MARKETER

Business at its best

COLUMNS

Code maintenance and data-syncing of distributed apps still massively inefficient

July 24, 2012

By Jan Rippingale

As a Gen-Xer who just turned 40, I never thought that I would sound like my "wisdom"-sharing Baby Boomer seniors, much less feel compelled to point out to the Gen Yers and Millennials that they are walking down roads fraught with already known and well-understood errors.

However, I cannot help but notice that the hopes and enthusiasm once felt for the PC as it permeated workplaces and homes, effectively distributing computer power to the next level of the "masses," mirror today's hopes and enthusiasm's for the mobile device as it effectively distributes computer power to the next level of the masses, even reaching the Second and Third World countries. The pitfalls and short-comings, however, are also mirrored.

The code maintenance and data-syncing required of distributed applications is still massively inefficient and ultimately unsustainable.

Somehow, no one seems to notice that we have already been through this learning curve. I propose a third, blended solution that raises the user's experience of flow about all other challenges, while also delivering the performance benefits of a client-side solution and the coherence of data and code maintainability of a server-side solution.

Framing the issue
The computer industry has experienced several pendulum swings between centralized and distributed architectures.

Mainframes were the solution in the 1950s and 60s, client-server solutions dominated in the 70s and 80s. Web-based solutions took over in the 90s to the present era, and now apps are fashionable.

The reasons for these shifts are clear in the advantages and disadvantages of each model. So, the trend and key technological turning points can be noticed and even predicted.

Centralized architectures such as mainframes and Web have a single data source and codebase. This makes maintenance as simple as it can be and eliminates data synchronizations.

Ironically, this efficiency contains the seed of its disadvantages.

Inevitably a centralized IT team will maintain the system and they will rationally prioritize their task lists and eventually level 4 priorities and below will become synonymous with "it will never happen."

At this point, users look for ways they can meet their own needs.

The other critical factor is how quickly the centralized repository can be accessed. Again, inevitably, system demand will grow such that what was perceived as unbelievably fast and efficient when compared with the familiar manual solution or fax-based solution, for instance will be perceived as too slow because it interrupts the user's

flow.

Ultimately, users forget the prior solution and begin comparing the system's speed with their own internal processing speed and every computer system built-to-date eventually fails against this measure.

Distributed architectures will then emerge as the solution to these shortcomings. Distributed architectures empower users to the next layer/level of users anyway to build their own solutions, effectively flipping the bird to the centralized IT team.

Some of these solutions will be impactful enough that it will be clear that centralized IT team missed a huge opportunity by not implementing this task and the limitations on creativity in the centralized model with become the new scapegoat and everyone who is anyone will begin to develop distributed solutions.

Also, since typically the distributed solution brings the computer power closer to the user via a PC or a mobile device, it runs faster for the user and they will expound on how much better they can work with reduced interruption of their flow.

Time will pass and soon the limitations of distributed architectures will rear their ugly heads.

Eventually, the maintenance required to distribute modified codebases a.k.a. your often more than weekly Microsoft updates will become annoying as it is a task that serves the computer but adds little or no perceived value to the user.

It will feel like busywork, at best, and an interruption of their flow, at worst, and they will begin to lean toward solutions that do not require this maintenance.

The distributed architecture's need for data synchronization leads to the same feelings if it runs perfectly and, if there are errors, it will be seen as a roadblock to clear communication and reliability.

Also, as soon as the computer power is distributed, it is outdated.

As the complexity of the distributed architecture increases, performance of the client-side solution will be perceived as slow, meaning it will not match the user's flow.

Put these code and data maintenance tasks together with the impossibility of evenly upgraded the distributed hardware and the matured centralized architecture becomes the new vogue. "Put it all up online. Let everyone see everything."

Again, some of the gains made will have such an impact that the overhead of distributed architectures will be the new scapegoat and everyone will focus on building in the new centralized architecture.

Is this beginning to sound familiar? Recognizable? Repeatable? I hope so, as the current computing ecosystem certainly has all of these historical dynamics and, yet, it also contains the opportunity to solve current limitations.

In the flow

Mainframes "taught away" form-distributed computing, so the next wave of a centralized architecture the Internet would have been correctly perceived as incredibly disjointed.

When anyone can have access to a centralized computer such as a Web server in this case, the ivory tower, rocket-science-level of exclusivity is lost in a relatively chaotic centralized solution.

The broad access to solutions, however, eventually missed the elegance, the quality inherent in the best-of-breed solutions and further consolidated cloud solutions began emerging.

Apps are the next client-side solution. They are fast, agile, and focused, enabling them to support micro-niches. They are also chaotic in that they are nearly impossible to maintain well in the complex ecosystem of mobile devices, and data synchronization is exceptionally painful.

So, rather than argue whether a centralized or distributed architecture is the right solution, it is likely more useful to outline the tensions that play against each other and then ask if there any new solutions to balance these tensions.

The first tension is "control." Who has it and why? This determines the rate and direction of change.

A smart, centralized group is efficient, but they are adverse to disruption and will ignore ideas with impact if they are too hard or too disruptive of the existing system or if the impact is too small for them to take notice from their elevated perspective.

The second tension is speed

The second tension is speed.

As obviously computers do calculations faster than most humans and yet people continually complain about slow computer systems.

The astute observer will notice that what a user means by speed is support of their flow, support of their thought processes for decision-making but also for their creative solutioning.

A deep consideration of what supports a human mind process and potentially restructure data into information can be quite valuable here (Tufte).

Ultimately, the solution to these tensions will have to be that the user is the one who is in "control" and the system speed needs to match or enable the user's flow. While a lot of progress has been made toward this ultimate goal, there is still much more to go.

A key insight overlooked here is that the user's view should remain the same until the user indicates they want it to change.

Rather than follow this perpetual pendulum swing, can we not find a third solution? It would need to maintain and support flow, perform well a typical client-side attribute and be easy to maintain, which is a typical server-side attribute.

First, everyone has to recognize that the user's flow is the key metric for determining usability. Anything less than perfect support of flow is an opportunity to improve or an opening to failure depending upon how fully embraced this truth is.

Currently, neither the client-side model nor the server-side model acknowledges this sufficiently.

Using the "navigation activator" to support flow, a user's experience on a page is never interrupted by a technical pause.

Everything the user wants to accomplish on a page is allowed to complete naturally and all the tasks are sent to the server when the user indicated that she is ready for a new page, meaning tasks are processed just prior to processing a navigation command, virtually invisible to the user. It is like magic, or as fast as thought, maximally supporting the user's natural flow.

Elegantly enough, by allowing the user to do as much on a page as they want, she experiences the performance benefits of a client-side solution because the server is not contacted until necessary as new information associated with a navigation has been requested.

Also, because the tasks are actually processed on the server, the heavy-lifting code is all maintained on the server, so the maintenance advantage of server-side solutions is also realized.

Technically, it is a win-win. The innovation driver, however, is the prioritization of the user's flow, prioritized above all convention.

Jan Rippingale is chief technology officer of i-Cue Design, a mobile commerce company in San Francisco. Reach her at jan@i-cuedesign.com.