**When is a cache not useful or even dangerous?**

*Performance tuning through cachine could be categorized into multi-layers:*

*Client-side (JS and CSS): Add an Expires or a Cache-Control Header will get it done for you. But note that there are more to do than only caching to enhance client-side performance. For detailes check Best Practices for Speeding Up Your Web Site*

*Server-side: this could be on many levels web server, scripting language, database, operating system, network, etc..*

*Good introduction and practical code examples can be found in Chapter 9 (Performance) - Developing Large Web Applications. It will talk about Caching CSS, Javascript, Modules, Pages, Ajax and Expire headers.*

*If we need to keep things simple on server-side do the following:*

*Install APC extension which will make PHP faster for you through the so called opcode caching. No special configuration, it will work silently for you.*

*Cache the full page for two-hours using this simple Pear library PEAR::Cache\_Lite.*

*For each database SELECT query cache the result in APC with a TTL of 5 Min, md5 hash the SELECT statement and use it as key for APC cache. Docs*

*In future if you have multiple servers and the performance becomes to be crucial before then you will need to look at:*

*Shared memory caching between servers. Check Memecache or even Membase*

*You need a reverse proxy solution: this basically layer between your user and server server so that it will serve the HTTP requests instead of your server. You can use for that Varnish, Squid or Apache Traffic Server.*

*Mysql innoDB engine is slow, you may need to go for faster engine such as XtraDB*

*Then maybe you will find that rational databases are stil slow for you. Then you will go for the key-value solution such as MongoDB.*

*Finally as references in web application performance check:*

*Front-end Performance: High Performance Web Sites, Even Faster Web Sites and High Performance JavaScript.*

*Back-end Performance: Pro PHP Application Performance and High Performance MySQL*

[*https://stackoverflow.com/questions/8082092/different-file-caching-methods-pros-cons*](https://stackoverflow.com/questions/8082092/different-file-caching-methods-pros-cons)

**Why does Event-Driven Architecture improve scalability?**

[**https://www.infoworld.com/article/3269207/enterprise-architecture/busting-event-driven-myths.html**](https://www.infoworld.com/article/3269207/enterprise-architecture/busting-event-driven-myths.html)

**What makes code readable?**

*Check out Jeff Atwood's Code Smells blog post. It pretty much sums it up. I'll add my personal ethos when it comes to good readable code:*

*Consistency: this applies to formatting, using braces, naming (variables, classes, methods) and directory layout (if you bury a source directory somewhere under /css I'm coming after you with a machete);*

*Size: if a function doesn't fit in its entirety on the screen in a normal IDE at a normal font size then you need a pretty darn good reason as to why not. Of course there are some valid cases for much longer functions but they are greatly outweighed by the egregious examples. Decompose as necessary to keep your functions simple;*

*Comment Judiciously: there is a tendency for some programmers to use comments as a substitute for readable code or to simply comment for the sake of commenting (like /\* finished \*/ comments right before return true;. Seriously, what's the point? Most (good) code explains itself;*

*Never Cut and Paste Within a Project: it's perfectly acceptable to take a code snippet from one project to another (every project is an island) but you should never take a non-trivial code segment from within one project to some other point within the project. Inevitably one changes and you leave some poor developer with the task of looking at these two or more code segments trying to work out how (and arguably more importantly, why) they are different; and*

*Avoid Repetitive Code: if you find yourself writing the same sequence of statements (or very similar) over and over again, abstract or parameterize it. If you see very similar statements the tendency is to skim over them assuming they're all the same (when typically they won't be in a way that matters).*

[*https://stackoverflow.com/questions/550861/improving-code-readability*](https://stackoverflow.com/questions/550861/improving-code-readability)

**What is the difference between emergent design and evolutionary architecture?**

**Leveraging reusable code, Part 2**

[**https://www.ibm.com/developerworks/java/library/j-eaed12/index.html**](https://www.ibm.com/developerworks/java/library/j-eaed12/index.html)

**Scale out vs scale up: how are they different? When to apply one, when the other?**

*Scaling up generally refers to purchasing and installing a more capable central control or piece of hardware. For example, when a project's input/output demands start to push against the limits of an individual server, a scaling up approach would be to buy a more capable server with more processing capacity and RAM.*

**How to deal with failover and user sessions?**

*You could set PHP to handle the sessions in the database, so all your servers share same session information as all servers use the same database for that.*

*A good tutorial for that can be found here.*

[*https://stackoverflow.com/questions/994935/php-sessions-in-a-load-balancing-cluster-how*](https://stackoverflow.com/questions/994935/php-sessions-in-a-load-balancing-cluster-how)

**What is CQRS (Command Query Responsibility Segregation)? How is it different from the oldest Command-Query Separation Principle?**

*Command and Query Responsibility Segregation (CQRS) is a pattern that segregates the operations that read data (queries) from the operations that update data (commands) by using separate interfaces. This means that the data models used for querying and updates are different.*

**How would you design a software system for scalability?**

*Scalability Design Principles*

*Avoid the single point of failure. ...*

*Scale horizontally, not vertically. ...*

*Push work as far away from the core as possible. ...*

*API first. ...*

*Cache everything, always. ...*

*Provide as fresh as needed data. ...*

*Design for maintenance and automation. ...*

*Asynchronous rather than synchronous.*

**Someone gave the name "The "C10k problem" to the problem of optimising network sockets to handle over 10.000 open connections at once. While handling 10.000 concurrent clients is not the same as handling 10.000 open connection, the context is similar. It's a tough challenge anyway, and no one is expected to know every single detail to solve it. It may be interesting to discuss the strategies you know to deal with that problem. Would you like to try?**

**PHP**

*For active connections (as in, loading and running a defined PHP script), yes, there will be as many PHP processes as active connections. But KeepAlive is about passive connections, and Nginx is very good at handling passive KeepAlive connections with very low resource usage - even for thousands of them.*

*The problem with Apache is that it, in the usual configuration with mod\_php and mpm\_prefork, needs a process for each connection even if it's just a passive KeepAlive. This means that most Apache servers in fact do need to have a PHP process in memory even if the connection is passive, but this is not the case if you run PHP as FastCGI. Apache can also handle lots of passive connections if you run PHP as FastCGI and choose the mpm\_worker which will create a more lightweight thread per connection, but it's still not as good as Nginx.*

[*https://stackoverflow.com/questions/16578874/what-specifically-makes-node-js-more-scalable-than-apache*](https://stackoverflow.com/questions/16578874/what-specifically-makes-node-js-more-scalable-than-apache)

**How would you design a decentralized (that is, with no central server) P2P system?**

*P2P or Peer-to-Peer is a very hard type of program to create, mostly because of its very structure. Most internet applications are Client-Server this is because a lot of headaches are solved just by having a fixed server you know you can connect to. At the end of the day, that's more or less all Napster did, it just indexed files and said who is currently hosting them.*

*The other problem with creating P2P software, is that developing it on your own, you will have very few peers to test with, even if you do create a bunch of virtual computers. You will find it hard to test it scaled to 00's of users.*

*First steps though, you will need to learn to program in a suitable language, something like C++ or C# maybe just make it as a console application to learn the technology.*

*Next, learn how to work with files. Not much use if you can't save stuff is it.*

*Networking next. Start with a client-server set up just to get to grips with transferring files. Make a server app that gives the files and a client app that downloads it. Then start to scale it to one server giving files to lots of clients.*

*Final step is to merge the client and server so that as the peer downloads more of the file, it can start to be a server and let other clients download from it.*

*If you want, now you can think about a GUI.*

**You may recall that Common Gateway Interface (CGI) is a standard protocol for web servers to execute programs (CGI scripts) that execute as Command-line programs on a server, and that dynamically generate HTML pages when invoked by a HTTP request. Perl and PHP used to be common languages for such scripts. In CGI, a HTTP request generally causes the invocation of a new process on the server, but FastCGI, SCGI and other approaches improved the mechanism, raising the performance, with techniques such as preforking processes. Can you imagine why has't CGI eventually win, and was instead replaced with other architectural approaches?**

*The questions is a little bit strange, it's not clear if you're asking only for PHP or your question is more general.*

*I will give you a brief overview from the PHP side.*

*First, the two protocols:*

*CGI scripts is a way how to run a server side script when a HTTP request comes; this has nothing to do with PHP*

*FastCGI is a "better CGI" - CGI is known to be slow, Fast CGI is a different approach with much faster results; this has also nothing to do with PHP.*

*Now the PHP related things:*

*mod\_php is running a PHP as Apache module - that is PHP request is run under Apache process with everything that goes with it - Apache processes are defined by Apache configuration, PHP is run with Apache permission etc.*

*PHP-FPM is PHP's FastCGI implementation; PHP-FPM runs as a standalone FastCGI server and Apache connects to the server using Apache's module, usually mod\_fcgid or mod\_fastcgi; I personally think this is much better than running as mod\_php, but it depends on your requirements and is also a little more complex; in this configuration, permission, processes related stuff & everything is run under PHP configuration, PHP user etc. & Apache connects to PHP as to a server; in this configuration it is also possible to have pool of PHP servers and to have PHP server on physically different machine than Apache. They say this is almost as fast as using Apache's module and there are benefits of better control over PHP configuration.*

*SuPHP - this was mostly used to address some problems of mod\_php related to permissions; with mod\_php PHP scripts are run under the Apache user/group; mod\_suphp can run the scripts in different user; I never used it, PHP-FPM now should be much better choice*

*So, basically:*

*CGI, Fast-CGI are protocols; CGI is slow, Fast-CGI is much faster*

*mod\_php (with underscore) and PHP-FPM are two basic ways how to run PHP*

*mod\_SuPHP is similar to mod\_php but can change the user/group that the process runs under*

[*https://serverfault.com/questions/645755/differences-and-dis-advanages-between-fast-cgi-cgi-mod-php-suphp-php-fpm*](https://serverfault.com/questions/645755/differences-and-dis-advanages-between-fast-cgi-cgi-mod-php-suphp-php-fpm)

**How would you defend the design of your systems against Vendor Lock-in?**

[**https://inviqa.com/blog/8-tips-for-avoiding-vendor-lock-in**](https://inviqa.com/blog/8-tips-for-avoiding-vendor-lock-in)

**What are the disadvantages of the Publish-Subscribe pattern at scale?**

[*https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c*](https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c)

**What's new in CPUs since the 80s, and how does it affect programming?**

[**https://danluu.com/new-cpu-features/**](https://danluu.com/new-cpu-features/)

**In which part of the lifecycle of a software performance should be taken in consideration, and how?**

[**https://airbrake.io/blog/sdlc/what-is-the-software-development-life-cycle**](https://airbrake.io/blog/sdlc/what-is-the-software-development-life-cycle)

**What’s the relationship between Performance and Scalability?**

[**https://blog.professorbeekums.com/performance-vs-scalability/**](https://blog.professorbeekums.com/performance-vs-scalability/)

**When is it OK (if ever) to use tight coupling?**

i0

A tightly coupled architecture integrates enterprise applications around a single point of truth, which is often a single spatially-enabled RDBMS. The types of applications that are linked include engineering design (CAD), facility records management (GIS), asset management, workflow, ERP, CRM, outage management, and other enterprise applications.

A major advantage of a tightly coupled architecture is that it enables the rapid and efficient processing of large volumes of data, provides a single point of truth instead of several, often redundant, data sources, and enables open access to data throughout the organization.

Tightly coupled architectures rely on standards such as SQL, ODBC, JDBC, and OLEDB, SQL/MM, and the Simple Feature Specification for SQL from the OGC, to provide open and secure access to data, including geo-spatial data, throughout the organization.

Loosely coupled Web services require substantial redundancies unlike tight coupling between clients and service, which minimizes redundancies.

One problem with asynchronous loosely coupled Web services is that for some business functions, it can exceed its resource capacity for the message queuing servers or system.

Loosely coupled Web services can be made to switch to tight coupling mode to avoid system overloads of scarce resources.

**What characteristic should a system have to be Cloud Ready?**

[**http://www.altivon.com/nine-characteristics-of-a-modern-cloud-architecture/**](http://www.altivon.com/nine-characteristics-of-a-modern-cloud-architecture/)

**Does unity of design imply an aristocracy of architects? Putting it simple: can good design emerge from a collective effort of all developers?**

**What's the difference between design, architecture, functionality and aesthetic? Discuss.**

*You're right yes. The architecture of a system is its 'skeleton'. It's the highest level of abstraction of a system. What kind of data storage is present, how do modules interact with each other, what recovery systems are in place. Just like design patterns, there are architectural patterns: MVC, 3-tier layered design, etc.*

*Software design is about designing the individual modules / components. What are the responsibilities, functions, of module x? Of class Y? What can it do, and what not? What design patterns can be used?*

*So in short, Software architecture is more about the design of the entire system, while software design emphasizes on module / component / class level.*

*In some descriptions of the SDLC (Software Development Life Cycle) they are interchangeable, but the consesus is that they are distinct. They are at the same time: different (1) stages, (2) areas of responsibility, and (3) levels of decision-making.*

*Architecture is the bigger picture: the choice of frameworks, languages, scope, goals, and high-level methodologies (Rational, waterfall, agile, etc.).*

*Design is the smaller picture: the plan for how code will be organized; how the contracts between different parts of the system will look; the ongoing implementation of the project's methodologies and goals. Specification are written during this stage.*

*These two stages will seem to blend together for different reasons.*

*Smaller projects often don't have enough scope to separate out planning into these to stages.*

*A project might be a part of a larger project, and hence parts of both stages are already decided. (There are already existing databases, conventions, standards, protocols, frameworks, reusable code, etc.)*

*Newer ways of thinking about the SDLC (see Agile methodologies) somewhat rearrange this traditional approach. Design (architecture to a lesser extent) takes place throughout the SDLC on purpose. There are often more iterations where the whole process happens over and over.*

*Software development is complicated and difficult to plan anyway, but clients/managers/salespeople usually make it harder by changing goals and requirements mid-stream. Design and even architectural decisions must bemade later in the project whether that is the plan or not.*

*Even if the stages or areas of responsibility blend together and happen all over the place, it is always good to know what level of decision-making is happening. (We could go on forever with this. I'm trying to keep it a summary.) I'll end with: Even if it seems your project has no formal architectural or design stage/AOR/documentaiton, it IS happening whether anyone is consciously doing it or not. If no one decides to do architecture, then a default one happens that is probably poor. Ditto for design. These concepts are almost more important if there are no formal stages representing them.*