**EMG 549 Intro to Software Development and Architecture for Engineering Managers Midterm Exam 11/10/2018**

Name: Ruchi Shah

Instructions:

Please complete all the questions and submit it in EMS

**0. Database Case Study 0**

**Review all 4 schema, tables, views, queries explained on following link and explain at least 5 unique points about each schema. Describe salient features of each schema, explain business being run using this schema and potential transaction scenarios. Give your comments on how each database is used in business in your own words.**

<https://www.ntu.edu.sg/home/ehchua/programming/sql/SampleDatabases.html>

**Ans:**

* **Employee Database:** 
  + The Database is about the about the employee with various elements described in the table
  + Employee is the primary table with primary key of the database is emp\_no INT.
  + The other tables are titles, salaries, dept\_manager and department and department employees.
  + Improvement can be done by merging all the department tables as we get an overview when we extract one table.
* **Salika (DVD Rental) Database**
  + The database is about getting information about DVD rental role around and inventory of the DVD
  + The primary keys in the database is film\_id SMALLINT, language\_id TINYINT, category\_id INYINT, actor\_id, staff\_id TINYINT, address\_id SMALLINT, city\_id SMALLINT
  + The database is very complex, and the database has too many primary keys, a single primary key is needed to cover the whole database.
* **Microsoft Northwind Trader**
  + The database is about Microsoft supply chain with shipper, customer, supplier and order.
  + The primary keys are OrderID INT, RegionID, TerritoryID VARCHAR, EmployeeID, SupplierID, CustomerID.
  + The database can be improved by removing many elements in the various characteristics in one table. There can be various sub tables that can be created to improve and find data.
* **Retailer Database** 
  + The database is about a retail chain with various turnarounds linked in various stages.
  + There is various database about employees, customers, offices, payments, order detail, products and product line
  + The database and database schema are clean and clearly mentions the data
  + Improvement can be done by adding view, procedure, function, trigger and event elements of the object store defined.

**1. a. Database Case Study 2**

**Please read through attached following case study: Refugees resettling in the U.S. in 2015: where from, where to and who?**

Case Study Link:

[https://www.dropbox.com/s/ckiazj9bgslfm00/Refugees%20resettling%20in%20the%20U.S.%20in%202015...%20from%2C%20where%20to%20and%20who%3F%20%E2%80%93%20Ben%20Collins.pdf?dl=0](https://www.dropbox.com/s/ckiazj9bgslfm00/Refugees%25252520resettling%25252520in%25252520the%25252520U.S.%25252520in%252525202015...%25252520from%2525252C%25252520where%25252520to%25252520and%25252520who%2525253F%25252520%252525E2%25252580%25252593%25252520Ben%25252520Collins.pdf?dl=0)

Please read the above case and explain to me what the problem, solution and summary of case is.

**Ans**:

* Problem: There were 2 data sets one which indicated nationality of the refugees and other indicated the religion. They wanted to create a data set when the both of the elements were combined
* Solution:
  + The collected all the raw data from arrivals\_destination\_nationality and rrivals\_nationality\_religion
  + They created 2 tables in destination and refugee
  + They created a primary key of origin and merged the 2 data sets
  + They now have a dataset where they get the destination, arrival and religion information in one table
  + They have used various group function and to merge the dataset and represent the data in geographical format as well.

**1.b. App development and Strategy Case Study HBR: Please summarize these cases in your own words and understanding in few paragraphs**

i. https://hbr.org/2015/11/the-changing-economics-of-app-development

ii. <https://hbr.org/2015/06/how-ibm-intuit-and-rich-products-became-more-customer-centric>

**Ans**: i. This article compares app market with the retail market at the grocery store where the name brands and popular items are added at the view range of the customer and makes the customer more likely to purchase. Same is in the world of apps where the popular ones are downloaded more than compared to the ones which are new. Apps are likely are to be purchased due to word of mouth and ads that run. The app’s most purchased are the done in in-built purchases within the app.

Paid promotion is one of the biggest ways to get mass downloads. The purchase of the drops 75% with 30 days and more 85% after 60 days.

ii. As it is rightly said that customer is king and even IBM is moving on that track where the customer is the king. They started by using the Waterfall method of product development where details and requirement of the product is been received from the customer. Later they have adopted Agile Methodology where the product was developed in 3 or 4 weeks.

Then they adopted design thinking approach where they used ethnography, anthropology, and other user-research techniques — putting users, rather than features, first in the planning process.

**2. Analyze attached Case Study on Using Python:**

**a. Read through the case study: Explain the problem at hand, solution presented, alternatives, your understanding of the case and summary.**

**b. Use Python code examples, database schema, database concepts to answer your question**

Case Study Link:

[https://www.dropbox.com/s/wopaaiowmsoqn8f/Python%20for%20Business%3A%20Identifying%20Duplicate%20Data%20-%2033%20Sticks.pdf?dl=0](https://www.dropbox.com/s/wopaaiowmsoqn8f/Python%25252520for%25252520Business%2525253A%25252520Identifying%25252520Duplicate%25252520Data%25252520-%2525252033%25252520Sticks.pdf?dl=0)

**Ans.** Python for business: Identifying Duplicate Data

Data Preparation is one of those critical tasks that most digital analysts take for granted as many of the analytics platforms we use take care of Wes Mc Kinney, author of Python for Data Analysis, defines Data Preparation as cleaning, munging, combining, normalizing, reshaping, slicing

The problem we have here is when we access the sales data, there are a lot of sales transactions that are duplicate

In the article the solution is presented in steps, they are as follows;

Import pandas library in python

Use pandas to import Excel file into python to make a DataFrame

To identify the duplicates in that DB add a column in DB as named is\_duplicated

Add a new boolean column to the DB

Sum up the Boolean to get a count of duplicate sales orders :df['is\_duplicated'].sum()

Qualify the impact of duplicate line items in terms of over counted units sold:df[df['is\_duplicated']].order\_item\_quantity.sum()

Use Vectorising method to multiply two columns and paste them into new columns.

Clean up the duplicate data:df\_nodup = df.loc[df['is\_duplicated'] == False]

Do a sanity check to make sure everything is in the right place:df\_nodup[df\_nodup['is\_duplicated']].line\_item\_total.sum()

Add the correct data and save it in a csv file :df\_nodup.to\_csv('2015sales\_nodup.csv', encoding='utf-8')

The case was about data cleaning and then adding the right data back to its place.

**3. Read through following case studies on following link and write atleast few paragraphs summary on any 3 cases. You may have to do more research to make sense of some topics or companies discussed. All cases analysis will get bonus points.**

<http://codesamplez.com/development/web-application-case-studies>

* Instagram: What powers Instagram
* Spotify: Backend Infrastructure at Spotify
* StackExchange: Stack Exchange’s High-Performance Server Stack
* Trello: Technology Stack at Trello
* Smashing Magazine: Improving Smashing Magazine’s Performance
* Github: Making MySQL Better at GitHub
* Symfony: Handling A billion requests a week with symfony2

Explain the problem being solved, technology being used to solve the problem and summarize the solution in your own words.

Ans: •**Instagram**: What powers Instagram

Speaking of Instagram, it has evolved in the just-over-1-year that we’ve been live to reach over 14 million users.

When it comes to choosing stacks, they follow some key principles: Keep it very simple; Don’t re-invent the wheel; Go with proven and solid technologies when you can.

From Operating System/Hosting, load balancing, application servers, data storage, task queue and push notification to monitoring.

**Operating System/Hosting** - Instagram use UbuntuL Linux 11.04 on Amazon EC2. Because at the time they only got 3 engineers, self-hosting wasn’t an option for them, they may revisit self-hosting in the future.

Load Balancing - They used to run 2 nginx machines and DNS Round-Robin between them. The downside of this approach is the time it takes for DNS to update in case one of the machines needs to get decomissioned.

So, they moved to Amazon’s Elastic Load Balancer, with 3 NGINX instances behind it that can be swapped in and out.

Application servers - application servers are usually used to handle our requests. They run Diangoon Amazon High-CPU Extra-Large machines.

They use http://gunicorn.org as their WSGI server, because it’s less CPU intensive. To run commands, they use Fabric.

**Data storage** - Most of Instagram data (users, photo metadata, tags, etc) lives in PostgreSQL. To get reasonable IO performance, they set up our EBS drives in a software RAID using mdadm. All of PostgreSQL instances run in a master-replica setup using Streaming Replication, and they use EBS snapshotting to take frequent backups of their systems. They use XFS as their file system, which lets the freeze & unfreeze the RAID arrays when snapshotting, in order to guarantee a consistent snapshot (our original inspiration came from ec2-consistent-snapshot.

**Task queue and push notification** - To notify real-time subscribers of an activity posted, they push that task into Gearman, which is a task queue system originally written at Danga. The most cost-effective solution they found was https://github.com/samuraisam/pyapns <http://t.umblr.com/redirect?z=http%3A%2F%2FPyAPNS&t=MzkzZTY0ZTcyYWE1NzFiNDFmMGExOTFlNjQ4YjJkZWZjNTk5MzE1MyxDTkhKTHlScA%3D%3D&b=t%3A3lU1XNmZZGkFDk-fPGXVAA&m=1>, an open source Twisted service that has handled over a billion push notifications.

**Monitoring -** they use Munin plugins, to graph metrics that aren’t system-level (for example, signups per minute, photos posted per second.

•**Spotify**: Backend Infrastructure at Spotify

Spotify grows fast, with number of daily users, the number of backend nodes that power the service, the number of hardware platforms clients run on, the number of development teams that work with products, the number of external apps they host on the platform, the number of songs they have in our catalogue.

As they grow, there are key components they need to be watching for their backend infrastructure.

**Speed** - they make great efforts to eliminate dependencies between teams and remove unnecessary complexity from our architecture.

**Autonomous squads** - each development team should always be able to move independently of other squads.

**Transparent code model** – This means that all code in the Spotify client, Spotify backend and Spotify infrastructure is available to all the developers at Spotify to read or change. If a squad is blocking on some other squad to make a change in some code, they always have the option to go ahead and make the change themselves.

**Open source** – Since Spotify is constantly pushing the scalability limits of the software, they use in backend they need to be able to improve the software they use in critical areas. They have contributed to many of the open source projects they use, for example Apache Cassandra and ZMQ. They use almost no proprietary software simply since we cannot trust that we will be able to tailor it to our ever-growing needs.

**Self-service infrastructure** – there is no need to wait for another team to get hardware, setup a storage cluster or do configuration changes. The Spotify backend infrastructure is built up of several layers of hardware and software, ranging from physical machines to messaging and storage solutions.

**Architecture** - The Spotify architecture partitions the problem in several different ways. Firstly, partitioning by features. A slightly oversimplified description is that all the physical screen area of all the pages and views in our clients is owned by some squad. All of the features in the Spotify clients belong to a specific squad. The squad is responsible for that feature across all platforms – all the way from how it appears on an iOS device or a browser via the real time requests. And these requests are handled by Spotify backend to the batch-oriented data crunching that takes place in our Hadoop cluster to power features.

**Backend Infrastructure**

**Storage** - Spotify set up a full storage cluster with replication and failover between sites is complicated so we are building infrastructure to setup and maintain multi-site Cassandra or PostgreSQL clusters as one unit.

**Message** - Spotify clients and backend services communicate with request-reply, messaging and pubsub. Spotify built its own low latency, low overhead messaging layer and are planning to extend it with high delivery guarantees.

**Capacity planning** - Spotify built an infrastructure that allows squads to scale their services automatically with load. Automatic scaling typically only works for bottlenecks that you are aware of, so there is always a certain level of human monitoring that the squad need to handle.

•**Smashing Magazine**: Improving Smashing Magazine’s Performance

During the years of its growth, Smashing Magazine has hit some major performance bottleneck.

Adopting responsive web design as a technique gets them a pretty bad rap for bloating websites but extremely difficult to maintain. Development and maintenance were taking a lot of time, the code base was full of minor and major fixes, and the infrastructure was becoming too slow. We ended up with a code base that had become bloated before the redesign was even released.

**Performance Issue** - Front End

Home page is loaded very slow. They wanted to create a remarkable reading experience on the website while avoiding the flash of unstyled text (FOUT), so web fonts were loaded in the header and, hence, were blocking the rendering of content. jQuery was required for ads to be displayed, and a few JavaScripts depended on jQuery, so they all were blocking rendering as well. Ads were loaded and rendered before the content to ensure that they appeared as quickly as possible. Images delivered by our ad partners were usually heavy and unoptimized, slowing down the page further.

**Performance Issue - Back End**

Because WordPress multi-install wasn’t available at the time, we ended up with six independent, autonomous WordPress installs with six independent, autonomous style sheets, which caused a problem, if an author wrote for two sections and we’d love to show their articles from both sections on one single author’s bio page? Well, we’d need to somehow pull articles from both installs and add redirects for each author’s page to that one unified page.

**Path tried**

**Performance Budget** - Different components require different amounts of time to load, yet some components of the page are more important than others. But the performance goals are - On Webpage Test <http://www.webpagetest.org/>, aim for a Speed Index <https://sites.google.com/a/webpagetest.org/docs/using-webpagetest/metrics/speed-index> value of under 1000; Ensure that all HTML, CSS and JavaScript fit within the first 14 KB.

They defined a performance budget that we set for the website — a threshold for what was acceptable. Admittedly, we didn’t concern ourselves with the start rendering time on different devices on various networks, mainly because we really wanted to push back as far as possible everything that isn’t required to start rendering the page, which means a Speed Index value that is way lower than the one we had set — as low as possible, actually — in all settings and on all connections, both shaky and stable, slow and fast.

Prioritization and separation of concerns - Concerns are separated into Core content. Essential HTML and CSS, usable non-JavaScript-enhanced experience; Enhancement. JavaScript, geolocation, touch support, enhanced CSS, web fonts, images, widgets; and Leftovers. Analytics, advertising, third-party content

By grouping content types into the same categories and identifying what’s critical, what’s important and what’s secondary. Fontdeck had server issues, the content didn’t appear at all, even though it was already sitting in the DOM, waiting to be displayed.

**Deferring Web Fonts** - they switched to Skolar for headings and Proxima Nova for body copy, delivered by Fontdeck. Overall, they had three fonts for each typeface — Regular, Italic and Bold — totaling in six font files to be delivered over the network because the new Smashing Magazine was all about beautiful, rich typography, not loading web fonts at all on mobile wasn’t really an option. The biggest problem they ran into was blocking rendering, even if the HTML, CSS and JavaScript had already loaded completely, the content wouldn’t appear until the fonts had loaded and rendered.

They looked into couple solutions

0.Typekit and Google’s WebFontLoader, which is an asynchronous script which gives you more granular control of what appears on the page while the fonts are being loaded.

0.Alternatives are @font-face directive only on larger screens by wrapping it in a media query, thus avoiding loading web fonts on mobile devices and in legacy browsers altogether.

0.The only other option was to improve the caching of fonts. They couldn’t do much with HTTP cache, but there was one option we hadn’t looked into: storing fonts in AppCache or local Storage.

Explain the problem being solved, technology being used to solve the problem and summarize the solution in your own words.

Ans:

**Instagram**

**Problem** - Amazon’s network disk system (EBS) doesn’t support enough disk seeks per second, so having all of our working set in memory is extremely important. Solution - To get reasonable IO performance, we set up our EBS drives in a software RAID using mdadm.

**Problem** - When there is no active memory profile already

**Solution** - vmtouch is a great tool for managing what data is in memory.

**Problem -** Redis helps power Instagram’s main feed, activity feed, and sessions system, but all of Redis’. Data needs to fit in memory.

Solution - Instagram team did was they ended up running several Quadruple Extra-Large Memory instances for Redis, too, and occasionally shard across a few Redis instances for any given subsystem.

**Spotify**

**Problem** - If there is a weak dependency between features, failure of one feature may sometimes lead to degradation of service of another feature, but not to the entire Spotify service failing.

**Solution** - Everything about one particular feature is concentrated to one squad it is very easy to A/B test features, look at the data collected and take an informed decision with all the relevant people involved.

Feature partitioning gives scalability, reliability and an efficient way of focusing team efforts.

And since all the users are not using all the features at the same time, the number of users that has to be handled by the backend of a particular feature is typically much smaller than the number of users of the entire Spotify service.

**Problem -** Building a storage solution for a feature that millions of people will use is not an easy task

**Solution -** For each feature the squad will have to create a storage solution that fits the needs of that particular service. The Spotify infrastructure offers a few different options for storage: Cassandra, PostgreSQL and Memcached.

If the feature’s data needs to be partitioned, then the squad has to implement the sharding themselves in their services

Setting up a full storage cluster with replication and failover between sites is complicated so we are building infrastructure to setup and maintain multi-site Cassandra or PostgreSQL clusters as one unit.

**Smashing Magazine**

During the years of its growth, SM has hit some major performance bottleneck.

Solution - work from Backend

To improve the performance, they first improved maintenance of their code. They did a code inventory where they looked thoroughly at every single class, ID and CSS selector. With a new CSS architecture, Grunt as a build tool and a few time-saving Grunt tasks, the task of maintaining the entire code base became much easier.

Then they cleaned up their back end. In the aim of achieving Multisite, Inpsyde is a good tool when they are restricted of creating a single WordPress installation from six separate installations. They first cleaned up the PHP templates, kicked unnecessary plugins into orbit, rewrote plugins they had to keep and added new ones where needed. And they merged all of the databases into one, and then created a single fresh and, most importantly, maintainable WordPress Multisite installation. The speed boost 400 to 500 milliseconds of improvement by avoiding sub-domain redirects and unifying the code base and the back-end code.

Solution - optimize Front end

They streamlined the classes for posts and pages, getting rid of all unneeded IDs, shortening selectors by lowering their specificity, and rooting out anything in the CSS we could live without crunched the CSS from 91 KB down to a mere 45 KB. And once CSS code base was in proper shape, they reconsider how assets are loaded on the page and how we can improve the start rendering time beyond having clean, well-structured code base. They focus on what matters most: the content, and the fastest way for readers to actually start reading our articles on their devices, define and refine the concept of the core experience on the website and use it as a baseline.

**4. Explain CAP Theorem, ACID a topics and BASE in terms of Database Management System concepts.**

**Ans. CAP Theorem:** CAP Theorem is a concept that a distributed database system can only have 2 of the 3: Consistency, Availability and Partition Tolerance. CAP Theorem is very important in the Big Data world, especially when we need to make tradeoffs between the three, based on our unique use case. On this blog, I will try to explain each of these concepts and the reasons for the tradeoff. I will avoid using specific examples as DBMS are rapidly evolving.

**ACID:** ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties of database transactions intended to guarantee validity even in the event of errors, power failures, etc.

**BASE:** The BASE acronym breaks down:

Basic Availability:

The database appears to work most of the time.

Soft-state:

Stores don’t have to be write-consistent, nor do different replicas have to be mutually consistent all the time.

Eventual consistency:

Stores exhibit consistency at some later point (e.g., lazily at read time).

**5. Please share with us best topics covered in the class at EMG549. Also suggest any other topics you would want to learn in this class that pertains to Software Development.**

**Ans. Best topic:**  Learning the backend of any companies running from OS to the database. Learning more about SQL database and basis of python.

**Suggestion:** Role of a manager and more management information.

**6. Please give your opinion about EMG 549 and give suggestion on how this course can be improved.**

**Ans. Best topic:**  Learning the backend of any companies running from OS to the database.

**Suggestion:** Role of a manager and more management information

**7. Please read the following article on Data Gravity and Distributed vs In Memory compute environment. Please summarize in your own words.**

<https://medium.com/@onix_systems/big-data-rivals-hadoop-mapreduce-vs-spark-c2f46402e2d4>

**Ans:** The article compares Apache Spark and Hadoop MapReduce as two big data processing frameworks to identify the right solution for a business. The author compares the two frameworks on the performance, processing functions, ease of use, fault tolerance, security, scalability, and cost.

Hadoop MapReduce provides more security and fault-tolerance than Apache Spark and is a good option for analysis on archived information or reporting purposes that do not need real time results. It is relatively easy to scale up to multiple nodes for data processing and does not need expensive RAM. Hadoop is natively java, is difficult to program, and needs tools like Pig and Hive for the interactive mode.

Apache Spark, on the other hand, is an advanced cluster computing engine that can process real-time as well as archived information. Its performance has been found to be 10x on disk and 100x in-memory. The user-friendly APIs support Scala natively as well as Java, Python and Spark SQL. It also provides an interactive shell that allows users to get prompt results.

8. Read the following case study on Canary Analysis, A/B testing and Blue Green Deployments. Explain each briefly

**Ans.** a) Canary:

1. It is one of the release strategies that can mitigate the risks of productions bugs that cannot be caught by other testing.

2. It is a technique used for version releases of the software, which mitigates or reduces the risk by gradually rolling out the changes to the subgroups.

3. The strategy of canary release is, before rolling it out to the change to entire/ platform, it is rolls out to small subset of users.

4. Canary deployment refers to a deployment done to a limited set of users who are not even aware if they are looking at new build. It is distributed to a small set of users.

5. It has mostly small impacts and in case any issues are found, they can be easily re-versed.

6. It proves beneficial for incremental code changes where code can be delivered quickly to evaluate whether or not the code release provides the desired outcome.

8 b) A/B testing:

1. A/B testing is different from blue green deployment, as A/B testing is mainly for measuring functionality in the app like usability, popularity, noticeability. Whereas, Blue green deployment is more about releasing new software safely and reducing the downtime.

2. A/B testing is usually for the UI parts of the app which would need backend support as well.

3. A/B testing is usually for comparing two versions of an app to check which one per-forms better.

4. A/B is also known as SPLIT testing as it is used to compare two versions or variations.

9. Read about top Python Frameworks for App deployment explain any 5 of them in detail.

<https://www.ongraph.com/a-list-of-top-10-python-frameworks-for-app-development/>

**Ans:** Top Python Frameworks:

1) Django:

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to re-invent the wheel. It’s free and open source.

* Django was designed to help developers take applications from concept to completion as quickly as possible.
* Django takes security seriously and helps developers avoid many common security mistakes
* Some of the busiest sites on the Web leverage Django’s ability to quickly and flexibly scale.

2) Turbo Gears:

TurboGears, born as a full stack layer on top of Pylons, is now a standalone WSGI web framework that can act both as a full stack framework (like Django) or as a micro framework (like Flask)

TurboGears, in its full stack mode, provides all the features you would require during development of a web application:

• Identification and Authentication

• Authorization

• Autogenerated Admin and CRUD

• Sessions

• Caching

• Schema Migrations

• Master/Slave Database Queries Balancing

• Request Bound Transactions

• Interactive Debugger

• Built in Profiling

• Pluggable Applications

It's also one of the few web frameworks officially supporting MongoDB as one of the primary storages backends, including support into the TurboGears Admin to autogenerate CRUDs from MongoDB models.

3) Pandas:

pandas are an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Pandas is quite a game changer when it comes to analyzing data with Python and it is one of the most preferred and widely used tools in data munging/wrangling if not THE most used one.

What’s cool about Pandas is that it takes data (like a CSV or TSV file, or a SQL data-base) and creates a Python object with rows and columns called data frame that looks very similar to table in a statistical software

4) Flask:

Flask is considered more Pythonic than the Django web framework because in common situations the equivalent Flask web application is more explicit. Flask is also easy to get started with as a beginner because there is little boilerplate code for getting a simple app up and running.

Flask was also written several years after Django and therefore learned from the Python community's reactions as the framework evolved. Jökull Sólberg wrote a great piece articulating to this effect in his experience switching between Flask and Django.

Flask's success created a lot of additional work in issue tickets and pull requests. Armin eventually created The Pallets Projects collection of open source code libraries after he had been managing Flask under his own GitHub account for several years.

5)Bottle:

Bottle is a fast, simple and lightweight WSGI micro web-framework for Python. It is distributed as a single file module and has no dependencies other than the Python Standard Library.

* Routing: Requests to function-call mapping with support for clean and dynamic URLs.
* Templates: Fast and pythonic built-in template engine and support for mako, jinja2 and cheetah templates.
* Utilities: Convenient access to form data, file uploads, cookies, headers and other HTTP-related metadata.
* Server: Built-in HTTP development server and support for paste, fapws3, bjoern, gae, cherrypy or any other WSGI capable HTTP server

10. Review following Software architectures and give examples of existing application employing these architectures.

**Ans**. N-tier Architecture: An N-Tier Application program is one that is distributed among three or more separate computers in a distributed network. A diagrammatic representation of an n-tier system depicts– presentation, application, and database layers.

These three layers can be further subdivided into different sub-layers depending on the requirements. Some of the popular sites who have applied this architecture are

MakeMyTrip.com

Salesforce enterprise application

Indian Railways – IRCTC

Amazon.com, etc.

The N-tier architecture helps to manage all the components (business layer, presentation layer, and database layer) of an application under one roof. Applications that uses small numbers of users on a local area network can benefit from n-tier architecture. Such architectural design ascertains maintaining, scaling up and deploying an application on the Internet efficiently.

Event Driven Architecture (EDS): Event-driven architecture (EDA) is a software architecture paradigm promoting the production, detection, consumption of, and reaction to events.

Example: The Java Swing API is based on an event driven architecture. This works particularly well with the motivation behind Swing to provide user interface related components and functionality. The API uses a nomenclature convention (e.g. "ActionListener" and "Action Event") to relate and organize event concerns. A class which needs to be aware of a particular event simply implements the appropriate listener, overrides the inherited methods, and is then added to the object that fires the event. A very simple example could be:

public class FooPanel extends JPanel implements ActionListener {

public FooPanel () {

super ();

JButton btn = new JButton ("Click Me!");

btn. addActionListener(this);

this.add(btn);

}

@Override

public void actionPerformed(ActionEvent ae) {

System.out.println("Button has been clicked!");

}

}

Microkernel Architecture :

The microkernel architecture pattern (sometimes referred to as the plug-in architecture pattern) is a natural pattern for implementing product-based applications.

Example : best example of the microkernel architecture is the Eclipse IDE. Downloading the basic Eclipse product provides you little more than a fancy editor. However, once you start adding plug-ins, it becomes a highly customizable and useful product.

Internet browsers are another common product example using the microkernel architecture: viewers and other plug-ins add additional capabilities that are not otherwise found in the basic browser (i.e., core system).

Microservices Architecture :

Microservice architecture is a form of service-oriented architecture (SOA) whereby software applications are built as a collection of loosely coupled services, as opposed to one monolithic software application. Each microservice can be created independently from the other, or even in a completely different programming language and run on their own.

Example : Netflix, eBay, Amazon, the UK Government Digital Service, Twitter, PayPal, The Guardian, and many other large-scale websites and applications have all evolved from monolithic to microservices architecture.

Space Based Architecture: is a software architecture pattern for achieving linear scalability of stateful, high-performance applications using the tuple space paradigm. (Wiki)

The SBA model is closely related to other patterns that have been proved successful in addressing the application scalability challenge, such as shared nothing architecture (SN), used by Google, Amazon.com and other well-known companies. The model has also been applied by many firms in the securities industry for implementing scalable electronic securities trading applications.