**Final Examination  
01219449 Principles of Software Architecture  
Department of Computer Engineering, Faculty of Engineering, Kasetsart University  
Tuesday 3rd May 2022: 13:00-24:00**

There are two parts in this final examination.

**Part A:** There are two main questions. You have the time from 13:00-16:00 to complete this part (120 points)

**Part B:** There is one task that you need to do, and you have time to submit it before midnight, Tuesday 3rd May 2022) (120 points)

**Instructions for Part A (13:00-16:00):** This part is an open-book and open-Internet examination. Computers, text books, lecture notes, and Internet communications are ALLOWED in this examination. But any types of verbal or non-verbal human communications are PROHIBITED during the examination time.  
  
Using the designated communication channel (i.e. Zoom), **you must open the video streaming of** **yourselves (your face)** with a proper dress **during the entire examination time.** You must set up your environment for this examination in a quiet room without any disruption or noise distraction.   
  
You must have **your student identification card** with you and prepare to show online at the beginning of the examination.

Read carefully and answer all questions in these sheets.

**1. [Bitcoin, Blockchain, Smart Contracts, Decentralized App (DApp)]**

1.1 A hash pointer is a kind of data structure that is used in the Bitcoin and blockchain application.  
  
If a hash pointer is

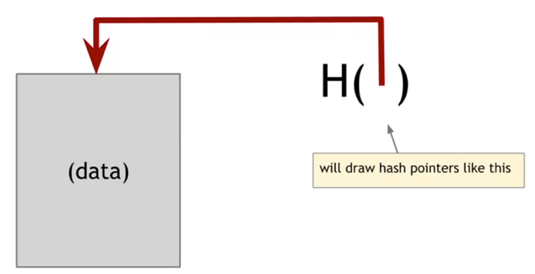
- a pointer to where some information is stored

- a cryptographic hash of that information.   
  
With the hash pointer, we can

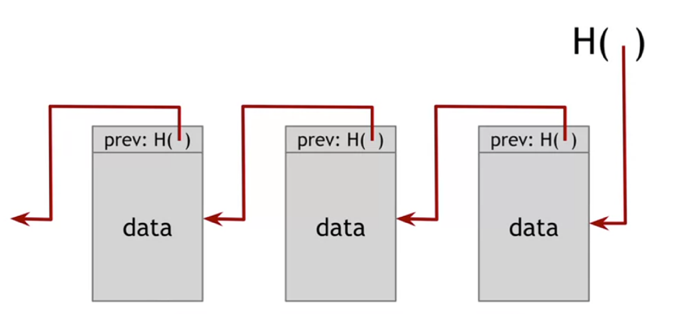
- ask to get the information back, and

- verify that it (the stored information) hasn’t changed.

The sentences in red above are represented with the figure below.



The figure below is a linked list where you have a series of blocks and each block has data as well as a pointer to the previous block in the list. The previous block pointer will be replaced with a hash pointer.

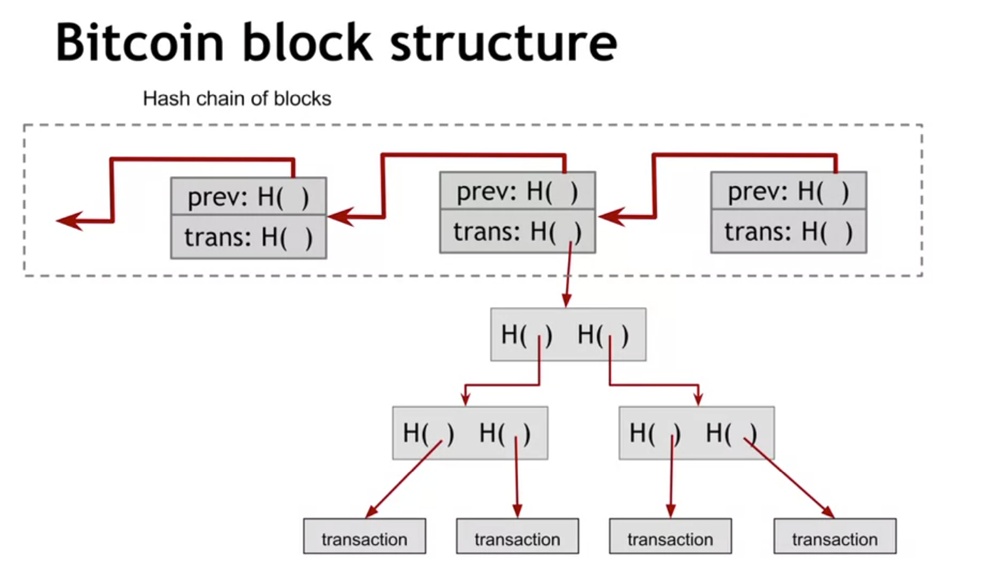


**Explain why this data structure is very useful as a temper-evidence log (why it is very difficult for any adversary to tamper with data later that's in the middle of the chain). (10 points)**

​Because in ​Blockchain network structure there is something calls “Merkle tree”.  Merkle tree is a binary tree formed by hash pointers. From given figure, each block is supposed to hold a certain number of transactions and transactions are grouped into pairs. The hash is computed for each pair and this is stored in the parent node, this structure continues till the root of the tree. So, According to the Merkle tree structure, it is very useful to tracking temper-evidence as log from root node and difficult to adversary between the leaf nodes since it grouped into pairs and alway checking each other.

1.2 For Bitcoin, transactions are actually grouped by blocks (that means each block in the Bitcoin blockchain contains not just one transaction but several transactions).

The figure below illustrates a data structure of the Bitcoin blockchain. It is a combination of two different hash-based data structures. On the top, we have a hash chain of blocks. Each one has a block header and then a pointer to some transaction data as well as a pointer to the previous block and the sequence, and remember these are hash pointers. And then we have a tree of all the transactions that are included in each block. So this is a hash tree or what is called a Merkle tree.



**1.2.1 Explain the design benefits of the above data structure. (20 points)**  
  
Just like I answered in the question 1.1., it is very useful to tracking temper-evidence as log from root node (Tampering Detection) and difficult to adversary between the leaf nodes since it grouped into pairs and alway checking each other (Efficient verification). And for other benefits, There is no delay in the transfer of data across the network, occupy less disk space when compared to other data structures, its root helps in making sure that the blocks sent across the network are whole and unaltered, and lastly it is the best solution if a comparison is done between the time complexity of searching a transaction in a block (using time = O(logn)).  
  
**1.2.2 Select the right answer - Blocks contain a tree of transactions instead of a flat list because   
(5 points)**

a. It results in smaller blocks

b. It's easier to insert or delete new transactions while the block is assembled

c. It enables efficiently proving that a transaction is included in a block.

1.3 Look at the following Solidity code.



**1.3.1 Describe the Solidity Code above, line by line, and its overall behavior. (20 points)**

Overall Behavior -> Voting System, Initial CadidateList (with Cadidate names) in Contructor, have functions to check if a cadidate in CadidateList or not, give a vote to a cadidate, and get the total vote of a cadidate.

Text

Description automatically generated

**1.3.2 Explain why we need ABI and Web3 library to develop the front-end DApp. (5 points)**

The Web3 has functions that to encode/decode parameters to ABI (Application Binary Interface). ABI has functions to call to the EVM (Ethereum Virtual Machine). EVM is a tool to executing smart contracts, transactions, and state. Which those executing are needed to built an application on a decentralized network that combines a smart contract and a frontend user interface.

**1.3.3 How can we list accounts in web3 1.x? (5 points)**

In web3-eth package that allows you to interact with Ethereum blockchain and smart contracts, there is a function called “getAccounts” that will return an array of addresses controlled by node.

Check-out -> [here!](https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html#getaccounts)

**1.3.4 What is the difference between .call and .send? (5 points)**

.send()

* will send a transaction to the smart contract and execute its method.
* It can alter the smart contract state.

.call()

* will call a constant method and execute its smart contract method in the EVM without sending any transaction. (No transaction = No gas fee)
* It cannot alter the smart contract state.

**1.3.5 Is it correct to send 1 ether like with .send({value:1})? (5 points)**

No, it isn’t.

The command “. send({value:1})” is mean sending 1 wei, since in smart contract transactions always work with wei unit. To send 1 eth, you will need to convert it to wei based on pool value, then receiver convert it back to eth.

**2. [MapReduce & Hadoop]**

**2.1 What are the differences between regular FileSystem and HDFS? (5 points)**

|  |  |
| --- | --- |
| **File System** | **Hadoop Distributed File System** |
| FS stores data as a single block. | HDFS is a commodity hardware, with multiple data blocks. |
| FS stores data in one location. | HDFS stores it into different DataNodes  (stores data across clusters) |
| FS uses Tree format to store data. | HDFS provides Master-Slave architecture for Data storage. (NameNode, DataNode) |
| FS has slow data retrieval, not appropriate for analysis of very big data. | HDFS has faster data retrieval, appropriate for big data. |
| FS is not fault-tolerant. | HDFS is fault-tolerant. |
| FS is cheaper because it does not needs extra memory for storing any data file. | HDFS is expensive because it needs extra memory to replicate the same data blocks. |
| In FS, we can access to data file directly. | In HDFS, we cannot access to data file directly. We need to require the where about of that data file from NameNode. |
| FS is less complex. | HDFS is more complex than FS. |

**2.2 Why is HDFS fault-tolerant? (5 points)**

Because it has a replication factor in nodes which means more than one copy of file piece (blocks) will be available in cluster when one is not reachable second copy can fulfill the request.

**2.3 Why NameNode (Master Service) should be run on the Enterprise-quality hardware, and DataNode should be run on commodity hardware (e.g. x86)? (5 points)**

Because NameNode not only stores metadata about all the blocks stored in HDFS, it also have to manage DataNodes and check their heartbeat signals. So, it needs high memory space and reliable performance hardware. On the other hand, DataNodes are the commodity hardware as it only need to store data like PCs or laptops with 128 MB as default. (ps. DataNodes are required in large numbers)

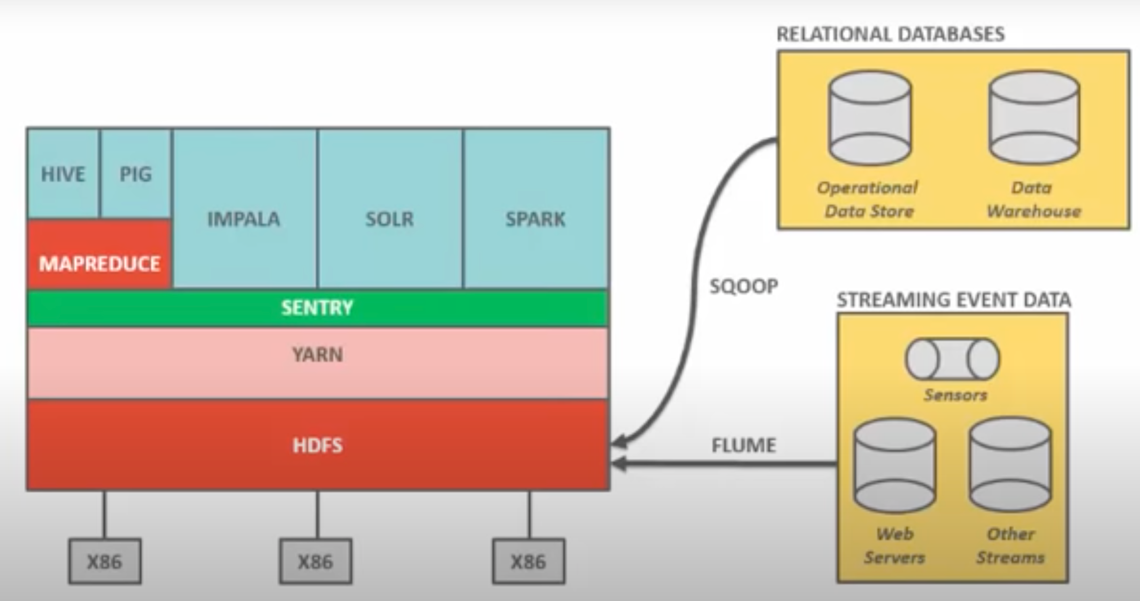
**2.4 If you have an input file of 350 MB, how many input splits would HDFS create and what would be the size of each input split? (5 points)**

* input splits to “3” nodes
* size of each input split to “128 MB, 128 MB, 94 MB”

**2.5 Why is MapReduce slower in processing data in comparison to other processing frameworks? (5 points)**

MapReduce is slower becaus, It is batch-oriented when it comes to processing data. During processing, whenever the mapper function delivers an output, it will be written to HDFS and the underlying disks. This data will be shuffled and sorted, and then be picked up for the reducing phase. The entire process of writing data to HDFS and retrieving it from HDFS makes MapReduce a lengthier process.

**2.6 Briefly describe key purposes of each component in the ecosystem below. (20 points)**



|  |  |
| --- | --- |
| HDFS | Store data |
| Sqoop | Tools use for pulling data from database |
| Flume | Framework for importing event based data |
| YARN | Resource manager |
| Sentry | A role-based authorization module for Hadoop. It provides the ability to control and enforce precise levels of privileges on data for authenticated users and applications on a Hadoop cluster. |
| Hive | Framework which allows users to read, write, and manage petabytes of data using SQL |
| Apache Pig | Tool which is used to process the large datasets |
| Impala | Parallel processing SQL query engine that runs on Apache Hadoop and use to process the data which stores in HDFS |
| Solr | Search server built on top of Apache Lucene, an open source, Java-based, information retrieval library –It is designed to drive powerful document retrieval applications. |
| Apache spark | An open-source distributed big data processing engine –It provides a common processing engine for both streaming and batch data. |

**Part B:** You have time to submit it before midnight on Tuesday 3rd May 2022. (120 points)

Record your own video presentation with your slides to explain about software design and development with Microservices and Domain-Driven Design (DDD).

- You can modify from the pdf file as attached below

- Using all key messages as explained in the VDO presentation by Chris Richardson, "Developing Microservices with DDD/Aggregates" for your presentation

https://www.youtube.com/watch?v=7kX3fs0pWwc

Or more explanation from   
https://www.infoq.com/articles/microservices-aggregates-events-cqrs-part-1-richardson/#:~:text=DDD%20Aggregates%20are%20the%20Building,Driven%20Design%20by%20Eric%20Evans  
  
and  
https://www.infoq.com/articles/microservices-aggregates-events-cqrs-part-2-richardson/

- Adding any messages and any more slides as you see appropriate, if you wish.

- Presenting and recording your VDO in English, putting it on a YouTube

**- Submitting the link of your YouTube video and your presentation file in our Google classroom (the Final Exam page)**

Topics of presentation that should be covered, at least the following

- Benefits of microservices (why more and more organizations are moving from monolithic to microservices, including also it's the future trend also for Thai organizations)

- Challenging issues to design and develop microservices

- Microservice Architecture (e.g. its key components)

- How API Gateway can act as the Facade (i.e. Facade is a design pattern)

- Domain Model

- ACID Transactions

- 2PC (Two-Phase Commit)

- What is CAP Theorem? (Trade off among Availability, Consistency, Partitioning)

- Domain-Driven Design, especially about Entities, Value Objects, and Aggregates

- What is Aggregate & Rules related to Aggregate

- How to maintaining consistency with Aggregates

- Event-driven architecture / event-driven, eventually consistent

- Rollback issue

- Dual Write Problem

- Event Sourcing, e.g. event stores

- CQRS

- Advantages and Drawbacks of Event Sourcing

- etc.