COMPSCI 630 Systems **Spring 2018**

Lecture 21

Lecturer: Emery Berger *Scribe: Qi Pan*

21.1 Vector clock review

We began by finishing up discussing what a vector clock is and that we should use it for Project 2. A vector clock is a generalization of Lamport clocks where essentially, you keep a "vector" of Lamport clocks for each process in your distributed system locally within a given process. The update rules are the same as Lamport timestamps but are applied to the vector element corresponding to the process in which an event occured. The rules can be loosely summarized as:

- If an internal event occurs in a process, increment the element corresponding that process by one
- If an external event occurs and a message is sent, the recieving process should update the element corresponding to the sending process to the max of their sent timestamp and the current local value of the timestamp

21.2 Bitcoin/Cryptocurrencies

The bulk of the lecture talked about how Bitcoin works and the various issues/consequences of it. First of all, Bitcoin is a decentralized online currency where the trust of the system relies on **proof of work** instead of a centralized trusted 3rd party which handles transactions.

Every user in the network has local copy of the ledger, a record of all transactions in the history of Bitcoin. To add a transaction to this ledger, one must find a nonce such that when appended to the updated ledger, creates a SHA256 hash lower than a certain number. Because SHA256 is a cryptographic function, it is one way meaning that the you cannot start with the number and get the associated nonce. Therefore, the only way to generate a valid nonce is through random guessing. By obtaining the correct nonce, you essentially provide proof that you've done a certain amount of computational work to update the ledger.

The ledger is actually not a static record, but it is a **block chain** of ledgers where each block contains a portion of the full ledger at a given time and they are chained by the previous hash. This means that the nonce you are trying to find is not just a function of the current ledger, but the previous successful hash as well. This allows the blockchain to be resistant to things like double spending as changing records in the past requires recomputing proper hashes for next block, and the next one, and so on.

Proof of work means that as long as a malicious party does not control $> 50\%$ of the computational power, then one can simply trust the longest chain being broadcasted around the network. In reality, users of the currency don't do the computation but leave that to miners who do the computation and get compensation for doing so. The monetary compensation is to motivate people to provide computation to the network to perform transactions.

21.3 Cryptocurrency pools

There are also pools of miners who share their computational resources and then split the generated revenue based on who has the most powerful machine. The problem from the managers point of view is how to know who has the most powerful machine. This is solved by adjusting shares of revenue based on successful "low" hashes or seeing how often each contributor gets close to a good hash. The incentive to join a pool is to get a steady rate of return and avoid burstiness in ones return.

There is also the problem of a pool gathering $> 50\%$ of the computational power and then as a group, maliciously attack the system. However, pools have an incentive to not do this and undermine the credibility of Bitcoin in order to maintain it's value and therefore the value they are mining for.

21.4 Cryptocurrency hardware and energy use

In order to compute random numbers faster, miners went from CPUs to GPUs and finally to specialized ASICs (Application Specific Integrated Circuit) meant to only guess nonces for SHA256 hashes. Using this hardware obviously consumes energy, and in order for the rate of return for miners to be worth it, it has to be higher than the energy cost required to get it.

Although ASICs have been optimized for SHA256 hashes, there are also memory hard functions such as scrypt which require large amounts of memory as well as compute speed. Since access to RAM is slow, this makes ASICs not much better at all than CPUs at computing a good hash. scrypt has been used in other cryptocurrncies like Dodgecoin and Monero.

21.5 Proof of stake

An alternative to proof of work is something called proof of stake which simply means your chances of being able to produce the next transaction is directly proportional to how much of the currncy you currently own. The same "50%" problem exists as in proof of work schemes but not it is if one party controls $> 50\%$ of the currency.

21.6 Cryptocurrency Attacks

An attack known as the **selfish mining attack** allows one to only need a third of the computational power to control the system rather than half. This is done by holding onto a successful hash, and calculating the next hash before telling the network of the first hash. Once both are found, once can dump both into the network. There is obviously a risk that someone else comes up with the first hash before you get your second one, but the math works out in expectation such that you only require a third of the computational power in order to control the whole network.

21.7 Legitimate use of blockchains

We briefly talked about the use of blockchain technology not just for cryptocurrencies, but for other things which can take advantage of its durability and how unforgable it is. One example was injecting sensitive

works into the blockchain to prevent censorship.

21.8 Dynamic and static analysis

The last topic we talked about was static and dynamic analysis relating to the two papers we read on Covery and Valgrind respectively. Briefly, static analysis tools look at compile time code to find errors while dynamic analysis tools do so on run time execution states. The line is a little blurred sometimes because of JIT compilers, which compiles code at runtime but in this lecture we focused on non-hybrid tools such as Covery and Valgrind.

Covery was a static analysis tool designed to catch bugs with a low false positive rate. This relates to what is generally known in information theory as the Precision-recall tradeoff which means that decreasing false positives (increasing precision) usually comes at the cost of increasing false negatives (decreasing recall). Covery explicitly ignored "soundness" to achieve low false positive rates, "soundness" refers to the fact that if the analysis reports no bugs, then there really are no bugs.

We described "soundness" in quotes because a static analysis tool can't always see the full picture or logic of the code. Take for example in C/C++ a pointer could be pointing to anything (a function) but the static analysis tool would not see this and account for the consequences of dereferencing it to call the mystery function. This was captured in a paper mentioned called "The Soundiness Manifesto" which suggests we should call this type of "soundness" as soundiness. Essentially, soundness works of of closed world assumptions (you know everything) but static analysis tools work in an open world where not everything is known about the functionality of the code.

We also discussed the DBI (dynamic binary instrumentation) framework Velgrind which allows dynamic analysis tools using shadow states to be easily created. These tools allow for a shadow state to be associated with any range of memory, an example is the taint bit for security. Tools created from Velgrind generally cause quite a bit of slow down (10x - 100x) and there are faster DBI's such as Pin but still create slow dynamic analysis tools on an absolute scale.