ECE 473/573
Cloud Computing and Cloud Native Systems
Lecture 24 Batch and Stream Processing II

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

November 13, 2024

Outline

Reading Assignment

- This lecture: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing http://people.csail.mit.edu/matei/papers/2012/nsdi_spark.pdf
- This and next lecture: Cloud Security

Outline

RDD Representation

- Each RDD consists of
  - Partitions as atomic piece of dataset.
  - Dependencies to parent RDDs.
  - A function to compute it from parent RDDs.
  - Metadata of partitioning scheme and data placement.
- Dependencies define communication needs.
  - Narrow dependency: each partition of the parent RDD is used by at most one partition of the child RDD, e.g. map and filter.
  - Wide dependency: multiple child partitions may depend on a single partition of the parent RDD, e.g. join and groupByKey.
  - Narrow dependencies allow for pipelined execution on a single node, elimating communication and simplifying fault recovery.
  - Wide dependencies require communications like MapReduce, and need complete re-execution for lost partitions.

Job Scheduling

- Since transformations are lazy, scheduling is trigger by actions.
- The scheduler will build a plan to compute the RDDs.
  - From RDD's lineage graph, as a directed acyclic graph (DAG) where vertices are partitions and edges are transformations.
  - The DAG is optimized by grouping vertices into stages, where within each stage transformations are merged, and no intermediate partitions are stored or communicated.
- The scheduler then decides what partitions are available and schedules tasks to compute missing partitions.
  - Follow the order of DAG to only schedule a task when all its input partitions become available.
  - Consider locality of data either in-memory or on-disk.
- Rerun failed task, persist RDDs to local drives if they require expensive communications to compute.

Memory Management

- RDD persistence options
  - In-memory storage as deserialized Java objects: fastest performance but large overhead in memory usage.
  - In-memory storage as serialized data: efficient memory usage.
  - On-disk storage: slowest, for RDDs too large to fit into memory, or too costly to recompute, usually due to expensive communication requirements from wide dependencies.
- Apply LRU policy to evict RDDs to make memory available.

Checkpointing

- While one can always recompute RDDs given their lineages, it is not efficient to do so for long lineage chains.
  - Specifically for wide dependencies within that require expensive communications.
- Checkpointing: store RDDs as output from wide dependencies on long lineage chains into stable storage.
  - Replicated as needed so no need to recompute if nodes fail.
  - RDDs are read-only so they can be written out in the background without impacting computation.
- Should checkpointing be specified by users or automatically decided?

Outline

RDD Implementation Details

Cryptography

CIA: Basic Components of (Computer Cyber) Security

- A king need to send messages to a general fighting in a war.
- Confidentiality
  - Only the king and the general can read the messages.
- Integrity
  - The general should only accept messages sent by the king.
- Availability
  - Some of the messages must be able to reach the general.

Additional Security Services

- Nonrepudiation: sender can not deny creation of message.
  - Can the general provide a proof to a third party that the command is from the King?
  - But who is the King?
- Authentication: who are you?
  - A.k.a. entity/user authentication, or identification
  - Within the context of computer cyber security, shall be built on top of a nonrepudiation service (but usually is not!).
- Access control/authorization: decide who can do what.

Symmetric Cryptography

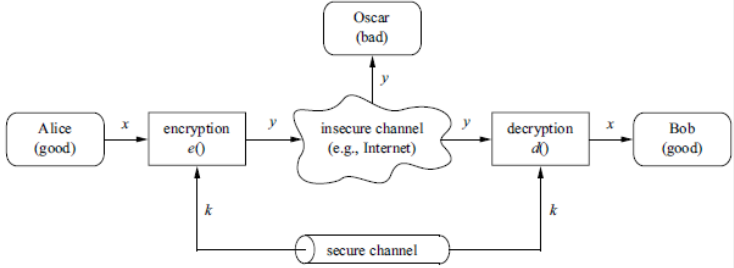

Fig. 1.5 Symmetric-key cryptosystem

(Paar and Pelzl)

- A mechanism for confidentiality
  - plaintext x, ciphertext y, and the key k
  - $e()$: encryption such that $y = e_k(x)$
  - $d()$: decryption such that $x = d_k(y)$
  - "Symmetric": both Alice and Bob know k.
- No "security by obscurity": Oscar knows everything except k

Hash Functions

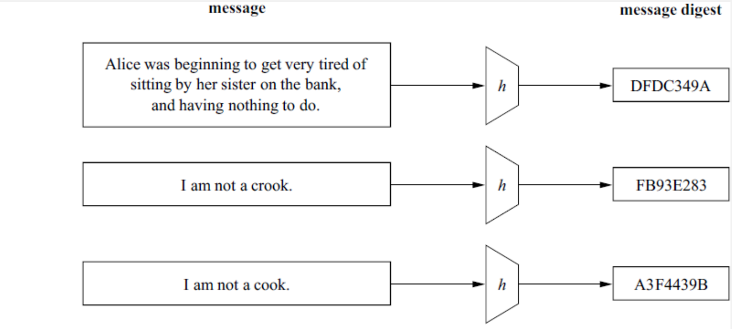

Fig. 11.3 Principal input–output behavior of hash functions

(Paar and Pelzl)

- Input x: messages of arbitrary lengths
- Output $z = h(x)$: message digest or hash, with fixed size.
- A strong hash function for use with cryptography prevents to find $x \neq x'$ such that $h(x) = h(x')$.

Authenticated Encryption with Associated Data (AEAD)

- Symmetric ciphers along cannot guarantee integrity.
- With the secret, hash functions can be augmented into message authentication code to validate integrity.
- Authenticated encryption combines the two to achieve both confidentiality and integrity.
- Very tricky to implement them together securely.
  - Use a well-defined AEAD algorithm like GCM, where software packages and hardware accelerations are widely available.
- AEAD cannot provide nonrepudiation service.
  - Neither Alice nor Bob can provide a proof that the message is encrypted by the other because they both know the secret.

Key Establishment

- To establishing a shared secret between two or more parties.
  - Which could be used later for AEAD.
- How can we solve this problem without a shared secret to begin with?

Public-Key Cryptography

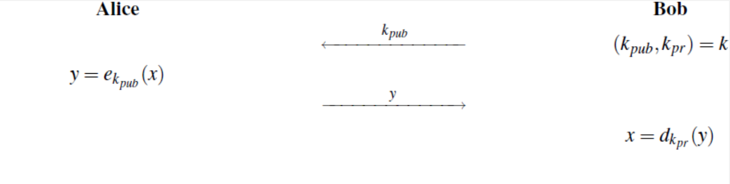

Fig. 6.4 Basic protocol for public-key encryption

(Paar and Pelzl)

- Key pair k: a public k_{pub} and a private (secret) k_{pr}.
  - No one should be able to derive k_{pr} from k_{pub}.
- Alice only need to obtain Bob's k_{pub} before they could share the secret x
- Such algorithms exist, e.g. RSA
- But how could Alice be sure that k_{pub} is from Bob?

Digital Signatures

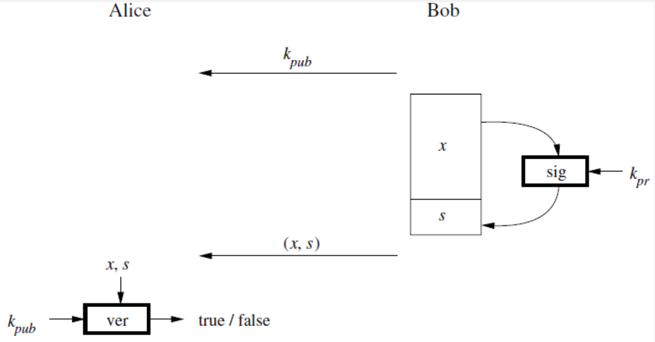

Fig. 10.1 Principle of digital signatures which involves signing and verifying a message

(Paar and Pelzl)

- Nonrepudiation: no shared secret
  - Bob signs with his private key k_{pr}.
  - Alice verifies with Bob's public key k_{pub}.
- Such algorithms exist, e.g. to run RSA reversely.
- Still, how could Alice be sure that k_{pub} is from Bob?

Public Key Infrastructure (PKI)

- A service to connect public keys to physical identities.
  - People, hosts, services, etc.
- Certificate Authority (CA): a trusted third-party.
  - Make use of public-key cryptography: $k_{pub,CA}$ and $k_{pr,CA}$.
  - For digital signatures only.
- Everyone knows $k_{pub,CA}$ to verify digital signatures from CA.
  - But how?
- How Bob proves to Alice $k_{pub,B}$ is from Bob?
  - Bob sends $k_{pub,B}$ to CA and ask CA to sign $(k_{pub,B}, ID_B)$.
  - CA returns Bob his certificate: $Cert_B = ((k_{pub,B}, ID_B), sig_{k_{pr,CA}}(k_{pub,B}, ID_B))$.
  - Bob presents Alice $Cert_B$ that Alice can verify with $k_{pub,CA}$.
- Authentication: in other words, Bob proves to Alice that he is Bob, with the help from CA.

Summary

- RDDs improve performance of distributed algorithms by making better use of local memory and CPUs to save on expensive disk and network I/Os.
- Public-key infrastructures combine symmetric cryptography and public-key cryptography to establish secure communication over insecure networks and to provide authentication.