Complex Systems Theory and Practice

Lecture 14

Piotr Syga

2022-01-21

Transactions

- during a single transaction we can use data from many nodes
- distributed transactions should be transparent for the user
- the distributed transaction is divided into local transaction based on the data allocation
- each local transaction should be treated as an atomic operation in a (larger) distributed transaction
- confirmation – 2PC

Two-phase commit (2PC)

How a simple transaction works?

- We start the transaction.
- We perform operations one-by-one without committing.
- If any of the operations fail we rollback to the state at the start of the transaction and abandon the query.
- We commit the operations.

What is the keypoint?

Knowing that some operation failed.

Distributed transactions

- We treat the local transactions as a 'simple' operations, yet we don't want to do them one by one, since the data is disjoint, we want to do them in parallel.
- Every local transaction may include different number of operations, hence they last various time.
- If a single local transaction fails, every local transaction should be rolled back, even those that managed to end already.
- How do we know that a node failed? What do we do when a node fails after their local transaction is done?

Two-phase commit (2PC)

Phase 1: Voting

1. The node to which the user is connected is chosen as a Coordinator and initiates the transaction.
2. The Coordinator sends a (local) query to each node and awaits the answers
3. Each participant performs the local transaction – if it succeeds, it sends `ack` message to the Coordinator, if the transaction fails it rolls back and sends `abort` message, if the node disconnects no message is sent

Two-phase commit (2PC)

Phase 2: Finalization

1. If any of the users sends `abort`, the Coordinator sends `ROLLBACK` message to all the nodes, every node that got the `ROLLBACK` message, rolls back their local transaction and sends `ack` to the coordinator. The Coordinator rolls back the distributed transaction only after receiving all the roll back acknowledgments.
2. We introduce a time limit after which the Coordinator decides that the node failed and if no messages are received in this time period, we proceed as with an `abort`. – *How to treat the timed out nodes?*
3. If every node acknowledged the execution of their transactions, the Coordinator sends `COMMIT` request to the nodes. All nodes should confirm it by sending `ack` to the Coorinator. After getting all the acknowledgments, the distributed transaction is committed. If some `acks` do not arrive, the transaction is rolled back just as in case of the `abort`.

In-doubt transactions

The transactions that have not been neither rolled back nor commited by the Coordinator are called *in-doubt* transactions.
To break the deadlock we may try several approaches:

- automatic (RECO) – transaction is delayed (cf. `sleep` command in OS), we try to repeat the operations after the node that did not respond with a message restablishes connection
- manual recovery:
  - we have to check which of the local transactions blocks the protocol (`transID`)
  - user with privileges to the transaction state perspective `DBA_2PC_PENDING`, and evocation graph `DBA_2PC_NEIGHBORS` runs: `SELECT * FROM SYS.DBA_2PC_PENDING WHERE local_tran_id = transID`
  - we have to check the `STATE` in the perspective (collecting, prepared, committed, *forced commit*, *forced abort*) and manually commit or abort the transaction:
    `COMMIT (ROLLBACK) FORCE transID`

Introduction

- Up to now we mentioned briefly that something is *time consuming* or *has higher complexity* – we want to formalize the notion.
- We can measure execution time of some function, however it does not show the scalability with the size of the data and with changing configuration of the machine running the code.
- We aim at presenting a mathematical representation of the time (and memory) required by a function, so that we may show scaling of the problem.
- In order to make our measure independent of the machine, we need to introduce a computation model.

Deterministic Turing Machine

DTM (standard notation)

An ordered tuple $M = (Q, \Sigma, \Gamma, \sqcup, \delta, S, F)$ is a deterministic Turing Machine if:

- Q – is a finite set of states,
- Σ – is a finite input alphabet,
- Γ – is a finite writing alphabet ($\Sigma \subseteq \Gamma$),
- $\sqcup$ – is a *blank* symbol that does not belong to Σ,
- δ – is a transition function $Q \times \Gamma \to Q \times \Gamma \times \{\rightarrow, -, \leftarrow\}$,
- S – is a selected element of Q that denotes the starting state,
- F – is a set of final (terminating) states.

Deterministic Turing Machine

DTM (simplified)

The tuple denoting DTM may be simplified to $M = (Q, \Sigma, \delta, S)$ if:

- Q – the set of states includes 3 dedicated elements `halt`, `yes`, `no`,
- Σ – the alphabet includes to additional symbols $\sqcup$ and $\triangleright$ (the latter may be skipped),
- δ – the transition function takes input from the whole alphabet: $Q \times \Sigma \to Q \times \Sigma \times \{\rightarrow, -, \leftarrow\}$,
- S – is a selected element of Q that denotes the starting state.

The state of DTM

Configuration

A triple (q, w, u) is called a configuration of a TM M, if $q \in Q$ indicates the current state of M, and the words $w, u \in \Sigma^*$ are located on the left (with the current symbol) and the right of the machine's header.

Transition between configurations

$(q, w, u) \xrightarrow{M} (q', w', u')$, $(q, w, u) \xrightarrow{M^k} (q', w', u')$,
$(q, w, u) \xrightarrow{M^*} (q', w', u')$

Example

$M = (\{s,\ q,\ q_0,\ q_1,\ q'_0,\ q'_1\},\ \{0,\ 1\},\ \delta,\ s)$
δ :

Q	0	1	$\triangleright$	$\sqcup$
s	$(q_0, \triangleright, \rightarrow)$	$(q_1, \triangleright, \rightarrow)$	$(s, \triangleright, \rightarrow)$	$(\text{yes}, \sqcup, -)$
q_0	$(q_0, 0, \rightarrow)$	$(q_0, 1, \rightarrow)$		$(q'_0, \sqcup, \leftarrow)$
q_1	$(q_1, 0, \rightarrow)$	$(q_1, 1, \rightarrow)$		$(q'_1, \sqcup, \leftarrow)$
q'_0	$(q, \sqcup, \leftarrow)$	$(\text{no}, 1, -)$	$(\text{yes}, \sqcup, \rightarrow)$	
q'_1	$(\text{no}, \sqcup, -)$	$(q, \sqcup, \leftarrow)$	$(\text{yes}, \sqcup, \rightarrow)$	
q	$(q, 0, \leftarrow)$	$(q, 1, \leftarrow)$	$(s, \triangleright, \rightarrow)$	

Example 2

$M = (\{s, q, q'\}, \{0, \#\}, \delta, s)$
δ :

Q	0	$\#$	$\triangleright$	$\sqcup$
s	$(s, 0, \rightarrow)$	$(q, 0, \rightarrow)$	$(s, \triangleright, \rightarrow)$	$(\text{halt}, \sqcup, -)$
q	$(q, 0, \rightarrow)$			$(q', \sqcup, \leftarrow)$
q'	$(\text{halt}, \sqcup, -)$			

Selected types of DTM

Some popular variants of DTM

- DTM with no $\triangleright$ ('two-way infinite tape')
- DTM with k tapes
- DTM with k writing alphabets
- DTM with input (read only), output (write only) and working (read and write) tapes
- DTM with 2D tape
- DTM with k headers

What we will use – a DTM with separate input and output tapes and k (sometimes $k = 1$) working tapes.

Equivalence between DTM and k-DTM

Time equivalence

For any DTM M with k tapes, that takes $f(n)$ transitions in order to complete the execution, we can construct a DTM M' with a single tape, such that $M(x) = M'(x)$ for each input and that requires at most $c\left(f(n)^2\right)$, for a constant c.

Time complexity

Halting time

If for a DTM M with k tapes a word x produces configuration sequence: $(s, \triangleright, x, \triangleright, \varepsilon, \ldots, \triangleright, \varepsilon) \xrightarrow{M^t} (Q_H, w_1, u_1, \ldots, w_k, u_k)$, for some $Q_H \in \{\texttt{halt}, \texttt{yes}, \texttt{no}\}$, we say that it stops in time t.

Complexity class

Let $L \subset (\Sigma \setminus \{\sqcup\})^*$ be a language decidable by a k tape DTM in such time that for each input x of length n, halting time $t < f(|x|)$. Then we say $L \in \mathrm{TIME}(f(n))$.

Linear speedup

Let $L \in \mathrm{TIME}(f(n))$, then for any $\varepsilon > 0$ $L \in \mathrm{TIME}(f'(n))$, for $f'(n) = \varepsilon f(n) + n + 2$.

Memory complexity

SPACE(·)

We say that a Turing Machine with input, output and k working tapes work in a memory limited by $f(|x|)$, if for any input word x the required memory $\sum_k |w_i u_i| \leq f(|x|)$.
If L is decidable in memory limited to $f(n)$, we say that $L \in \mathrm{SPACE}(f(n))$.

Proper complexity functions

Let $f : \mathbb{N} \to \mathbb{N}$. We call f a proper complexity function if:

- f is non decreasing
- there exists a k-DTM M with input and output tape that can calculate $f(|x|)$ in unary coding
- M stops after $O(|x| + f(|x|))$ steps and uses $O(f(|x|))$ memory.

Nondeterministic Turing Machine

NTM (simplified)

A nondeterministic Turing Machine N is represented by a tuple $N = (Q, \Sigma, \Delta, S)$ where:

- Q – is a finite set of states (additionally we have final states Q_H: `halt`, `yes`, `no`) ,
- Σ – is a finite alphabet (additionally we have symbols outside of Σ: $\sqcup$, $\triangleright$),
- S – is a starting state from Q,
- Δ – is a relation that is a subset of $(Q \times \Sigma) \times ((Q \cup Q_H) \times \Sigma \times \{\leftarrow, -, \rightarrow\})$.

The nondeterministic state

Configuration

A triple (q, w, u) we call a configuration of NTM N, if $q \in Q$ indicates the current state of N, and $w, u \in \Sigma^*$ are on the left and on the right, respectively, of the machine's header.

Transition between the configurations

The transition between the configurations is not a function, hence we can move to multiple configuration *at the same time*. We may follow all the paths or (nondeterministically) select a single one.

Example

The language of all composite numbers

$L = \{n : \exists_{1<k<n}\ k \mid n\}$

- we select (using nondeterminism) a calculation path that leads to termination („we guess" k)
- in a deterministic fashion we divide n by k
- if the reminder is equal to 0: `yes`, otherwise `no`

Example

The language of all composite numbers

$L = \{n : \exists_{1<k<n}\ k \mid n\}$

- we select (using nondeterminism) a calculation path that leads to termination („we guess" k)
- in a deterministic fashion we divide n by k
- if the reminder is equal to 0: `yes`, otherwise `no`

Alternatively:

- we select (using nondeterminism) a calculation path that leads to termination („we guess" k_1 and k_2)
- in a deterministic fashion we multiply $k_1 \cdot k_2$
- if $k_1 \cdot k_2 = n$: `yes`, otherwise `no`

Time complexity

Halting time

We say that NTM N decides the language L in time $f(n)$ if for any $x \in L$ of length n, there is a path such that $(q_0, \triangleright, x) \xrightarrow{N^k} (\mathtt{yes}, w, u)$ and $k \leq f(|x|)$.

Complexity class

The set of languages decidable by NTM in $f(n)$ is a complexity class $\mathrm{NTIME}(f(n))$. We may show that $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$.