Architektura a komunikace v distribuovaných systémech

Paralelní a distribuované systémy, přednáška 5

Tomáš Urbanec

Katedra informatiky PřF UPOL

4.11.2026

Co nás čeká?

1. Úvod do distribuovaných systémů
   - Co je to distribuovaný systém?
   - Architektury DS.
2. Komunikace v distribuovaném systému
   - Opakování ISO/OSI a TCP/IP
   - Remote Procedure Call (RPC)
   - Message-Oriented Communication (vzory, MOM)
   - Inter-Process Communication (IPC)
   - Multicast a broadcast

Distribuované systémy

Paralelní vs. distribuovaný systém

- Distribuovaný systém lze chápat jako speciální případ paralelního systému.
- Více (autonomních) systémů spolupracujících na jednom úkolu.
- Zvenku se jeví jako jeden systém.
- Mohou být na jednom uzlu/stroji, na LAN i WAN.
- Může to být více distribuovaných systémů tvářících se jako jeden (větší) systém.
- Rozdíly oproti dosud uvažovaným paralelním systémům:
  - Absence sdílené paměti.
  - Komunikace (po síti? spolehlivost? rychlost? latence?).
  - Absence synchronizace času.
  - Problémy jsou obvykle složitější.
  - ...

Omyly distribuovaného výpočtu

Fallacies of distributed computing [1]

1. Síť je spolehlivá.
2. Latence je nulová.
3. Přenosová kapacita je neomezená.
4. Síť je bezpečná.
5. Topologie se nemění.
6. Existuje jeden administrátor.
7. Přenos dat nic nestojí.
8. Síť je homogenní.

Příklady

Distribuované systémy

- Aplikace komunikující s databází (frontend, backend, DB, ...).
- Portál, STAG, ...
- Steam, WoW, LoL, CS, Valorant, ...
- Web
- IoT
- Blockchain
- Internet

Architektura distribuovaných systémů

Architektura DS

- software vs. systémová architektura
  - Jak vypadá struktura a rozhraní jednotlivých komponent?
  - Jsou provázané?
  - Jak budou komponenty rozloženy do sítě?
- Mnoho různých možností (dále)
- Často se v nich prolínají oba pohledy.

Klient/Server

Architektura DS

- Klienti (uživatelé, aplikace, . . .) požadují službu od serverů. Servery služby nabízejí.
- Jeden uzel může být pro někoho server a pro někoho klient.
- Typicky (ale ne nutně) po síti.
- Centralizované okolo serveru.
- Klienti typicky jednoduché procesy, řeší zejména interakci se serverem (API).
- Request-response (požadavek-odpověď) komunikace. Často asynchronní.
- Př. webové služby, souborové servery, databázové servery, emailové servery, . . .

Peer-to-Peer (P2P)

Architektura DS

- ≈ Extrémní verze myšlenky „uzel může být klient i server“
- Každý uzel má stejnou roli i odpovědnost.
- Každý uzel zná své sousedy.
- Plně decentralizované.
- Snadno škálovatelné.
- Velmi dynamické (uzly mizí a přibývají dle libosti).
- Př. sdílení souborů, Blockchain, BitTorrent (?)

Vrstvená architektura

Architektura DS

= Layered

- Systém je rozdělený do logických vrstev.
- Velmi často 3 vrstvy:
  - Uživatelské rozhraní
  - Logika aplikace
  - Datová vrstva
- Vrstvy odděleny (nezávislé) a komunikují mezi sebou (API).
- Komunikace typicky jen se sousedními vrstvami (ne vždy).
- Modulární, *teoreticky* dobře udržovatelné.
- Př. typická větší webová aplikace (FE v JS, BE v ?, data v databázi/ích)

Architektura zaměřená na služby

Architektura DS

= Service oriented architecture

- Snaha o znovupoužitelnost komponent jako služeb.
- Služby mohou běžet na různých strojích, vystavují rozhraní (API).
- Vysoká modularita.
- Dobrá škálovatelnost.

Mikroslužby

Architektura DS

= Microservice

≈ Neprovázaná SOA

- Jednotlivé mikroslužby by mělo být možné nasazovat nezávisle.
- Obecná SOA může vyžadovat nasazení/úpravu více služeb najednou (provázanost).

- Členěné spíše podle business logiky než podle klasických vrstev či technologií.

Architektura zaměřená na události

Architektura DS

= Event based architecture / Event driven architecture / Publish-subscribe

- Centrální prvek jsou události a jejich průběh systémem.
- Komponenty asynchronně produkují a konzumují události.
- Komponenty na sobě plně nezávislé.
- Postaveno okolo systému, který události sbírá a distribuuje.
- Př. chytrá domácnost, obecně IoT, Kafka...

Hybridní architektura a další

Architektura DS

- Hybridní = kombinace více přístupů
- Př. BitTorrent - P2P s prominentními uzly majícími roli serverů
- Př. Větší systém, používající microservice pro pomocné služby, client-server pro komunikaci s uživatelem a P2P pro sdílení obsahu mezi uživateli.
- Další:
  - Message oriented,
  - Leader-Follower (Main-replica),
  - Middleware,
  - ...

Komunikace v DS

Komunikace v distribuovaném systému

- Nemáme sdílenou paměť.
- Musíme komunikovat posíláním zpráv.
- To lze provádět na různých úrovních abstrakce.
  - Budování zprávy bit po bitu,
  - ...
  - Transparentní volání metod instancí objektů na jiném stroji.
- Reálně máme vrstvený model ve stylu ISO/OSI a nad ním abstrakce.

ISO OSI a TCP/IP

Opakování síťové komunikace

- ISO/OSI jako teoretický rámec.
- Dva systémy komunikují pomocí zpráv.
- Zprávy mají jasný formát, obsah i význam.
- Formalizováno komunikačním protokolem.
- Poskytují komunikační služby.
- S navázáním spojení či bez něj.
- Založeno na vrstvách (tabule).
- Zpráva zabalená/rozbalená po vrstvách (tabule).
- Reálně TCP/IP.
  - Aplikační,
  - Transportní - TCP/UDP,
  - Síťová - IP,
  - Síťové rozhraní.

Middleware protokoly

Síťová komunikace

- Další služby/vrstvy/protokoly čistě v aplikační vrstvě.
  - DNS - překlady doménových jmen na adresy.
  - Autentizační/autorizační protokoly.
  - Atomické transakce (distributed commit protocol).
  - Vzájemné vyloučení na aplikační vrstvě.
  - Komunikační protokoly - RPC, AMQP, ...
  - ...
- Typicky sdílené mnoha aplikacemi.
- Nás zajímá právě komunikace na aplikační vrstvě.
- Tj. fyzický přenos informace necháme na OS.

Typy komunikace

Middleware protokoly

- Middleware zprávu
  - drží, dokud není doručena → perzistentní.
  - pouze přenese do cíle → dočasná/tranzientní.
- Odesílatel
  - jen odešle a pracuje si dál → asynchronní.
  - čeká na potvrzení převzetí middlewarem
  - čeká na potvrzení přijetí příjemcem
  - čeká na potvrzení zpracování příjemcem
  - → synchronní
- Různé kombinace
  - tranzientní s potvrzením zpracování → RPC
  - perzistentní s potvrzením převzetí → message-queue/broker
  - ...

Remote Procedure Call

- Proč nám nestačí prostě odeslat/přijmout zprávu?
- Transparentní komunikace vs. technické detaily.
- Remote procedure call - můžeme volat přímo metody/procedury dostupné na druhé straně.
- *A* zavolá proceduru/metodu *B*, předá jí parametry a dostane výsledek.
- Z pohledu *A* stejné jako při volání lokálních procedur/metod.
- *A* neřeší žádné zprávy.
- Myšlenka jasná, provedení náročnější . . .
  - *A* a *B* mohou být na různých strojích.
  - Mají různé adresní prostory (s různým obsahem paměti).
  - Nějak musíme předat argumenty a výsledek.
  - Mohou mít jiné konvence (architektura, endianita, . . .)
  - *B* může explodovat (nebo jen vypadnout ze sítě) . . . před zavoláním/po zavolání ale před odpovědí. . .

→ Úplná transparentnost není možná: latence, paměť, částečná selhání, souběžnost [2].

Idea RPC

Pohled klienta

- Volající netuší, že volá proceduru vzdáleného objektu (ani nechce tušit).
- Z jeho pohledu normální lokální volání a čeká na výsledek.
- Middleware volajícímu předhodí obrys vzdáleného objektu - *stub*.
- *A* vyvolá metodu stubu $\rightarrow$ middleware zařídí komunikaci
  1. Vytvoří zprávu s požadavkem,
  2. včetně argumentů
  3. vykomunikuje s OS odeslání zprávy serveru (*B*),
  4. počká na odpověď,
  5. rozbalí odpověď,
  6. vrátí výsledek volajícímu.
     - Vše transparentní.

Idea RPC

Pohled serveru

- Server ví, co nabízí světu.
- Vše, co nabízí, má svůj stub.
- Skutečný objekt netuší, že jeho metoda byla vyvolána vzdáleně.
- Přijde volání metody M objektu $O \rightarrow$ middleware zařídí komunikaci
  1. Přijme od OS zprávu s požadavkem *O.M(args)*,
  2. předá ji stubu O,
  3. stub O rozbalí argumenty *args*,
  4. vyvolá metodu M objektu O s argumenty,
  5. počká na výsledek metody,
  6. zabalí odpověď,
  7. vykomunikuje s OS odeslání odpovědi klientovi (A).
     - Vše transparentní.

Problémy RPC

- Předávání parametrů - marshalling/unmarshalling
- Zpráva je posloupnost bytů - jak ji interpretovat?
- Endianita atd.?
- Vše musí být jasné předem
  - Nezávislý (na stroji i síti) formát komunikace,
  - Podpora v jazyce,
  - Rutiny pro převod z nezávislého formátu na formát stroje a naopak.
- Typ parametru
  - jednoduchý typ?
  - referenční typ (tj. ukazatel)?
  - složitý referenční typ?

→ automatický marshalling?

→ globální reference? (např. sdílený souborový systém)

Realizace RPC

Jak to tedy udělat?

- Podpora RPC v jazyce (např. Java RMI)
- Nezávislý RPC protokol
  - Formát zpráv,
  - Reprezentace datových typů,
  - Popis rozhraní dostupných objektů
    - Interface definition language (IDL)
    - Stuby vygenerované pro klienty i server
    - Klidně v různých jazycích
  - XML-RPC/SOAP, JSON-RPC, gRPC

Varianty RPC

- Klient při zavolání procedury čeká na výsledek (synchronní přístup).
- To může být plýtvání zdroji.
- Asynchronní RPC
  - Server potvrdí přijetí požadavku.
  - Klient nečeká a pracuje dále (callback, čekající vlákno, ...)
- Jednosměrné RPC
  - Klient jen zavolá proceduru.
  - Neřeší se žádné potvrzování.
  - Spolehlivost spojení?
- Multicast RPC
  - Prostředek ke škálování.

Message-Oriented Communication

- RPC je hezky transparentní, ale občas nevhodné
  - Komunikující strany fungují jindy (nezávisle)
  - Obecný tlak RPC na synchronní komunikaci.
  - Komplikovaná implementace.
  - Obecné zprávy v systému vs. předdefinované možnosti v RPC.
- MOC - Obecné posílání zpráv
- MOM - Message-Oriented Middleware
- Perzistence zpráv?
- Modely komunikace
  - Request-Reply
  - Publish-subscribe
  - Pipeline
- MPI - Message Passing Interface - standardizace pro tranzientní zprávy

Message-Oriented Communication

Základní přístupy

- Roura (pipe)
  - Jednosměrné spojení výstupu procesu a vstupu jiného procesu.
  - FIFO
  - Unixová roura, channel v Go, Pipe v Pythonu, ...
  - Unix: | (ps aux | grep grep)
  - Unix: Nepojmenované (dočasné) a pojmenované (soubor? oboustranné? perzistence?)
- Socket
  - Abstrakce portu
  - Operace klienta: connect, send, receive, close
  - Operace serveru: bind, listen, accept, send, receive, close
  - Pojem websocket?
    - Podobná myšlenka, ale vyšší úroveň
    - Nad HTTP
    - $\approx$ povýšené HTTP spojení

Message-Oriented Communication

Základní přístupy

- Fronta zpráv (message queue)
  - Centrální místo pro výměnu zpráv
  - Zprávy daného typu
  - Perzistentní, oboustranné
  - Srovnání s pojmenovanou rourou?
  - Zprávu zpracuje jeden konzument, po potvrzení zmizí
  - Může být komplikované → standardizace AMQP
  - Př. RabbitMQ, Amazon SQS
- Message broker
  - Obecné zprávy (procesy více typů)
  - Rozumí zprávám: směruje, konvertuje, kategorizuje, ...
  - Lze přidat nové typy (plugin)
  - Systém topiců (publish-subscribe model)
  - Př. RabbitMQ (exchange), ActiveMQ
  - Apache Kafka: spíš distribuovaný log, zprávy po přečtení nemaže, konzument si drží pozici (offset)

Inter-Process Communication

- Komunikace mezi procesy na stejném stroji
- Lze uvažovat vše zmíněné
  - Sdílená paměť
  - Mutexy, semafory, ...
  - Roury, fronty, ...
  - RPC (ale proč?)
  - Většinou není perzistentní.
  - Většinou nejsou problémy sítě.
  - Procesy mohou sdílet stav.

Multicast

- Zasílání zpráv více uzlům
- Broadcast = multicast všem uzlům
- Overlay (struktura nad sítí)
  - strom (unikátní cesta)
  - mesh (více cest, robustnost)
- Hodně zpráv $\rightarrow$ přetížené spoje
- Více typů
  - Flooding (broadcast)
    - uzel přeposílá všem (kromě zdroje)
    - lze udělat více overlayů (podle skupin)
  - Epidemické protokoly/gossip
    - paralela k virům/drbům
    - snažíme se infikovat/informovat své okolí
    - náhodný výběr cílů
    - push/pull přístup

Literatura I



A. Rotem-Gal-Oz: *Fallacies of Distributed Computing Explained*. `https://lasr.cs.ucla.edu/classes/188_winter15/readings/fallacies.pdf`. Citováno 4.11.2026.



J. Waldo, G. Wyant, A. Wollrath, S. Kendall: *A Note on Distributed Computing*. Sun Microsystems Laboratories, SMLI TR-94-29, 1994. `https://mirror.cs.odu.edu/kde/devel/smli_tr-94-29.pdf`. Citováno 4.11.2026.

Changelog