

Komunikace v distribuovaném systému

Paralelní a distribuované systémy, přednáška 6

Tomáš Urbanec

Katedra informatiky PŘF UPOL

4.11.2025

Co nás čeká?

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

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 za pomoci 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/transientní.
- 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
 - transientní 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 ji 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...

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 → 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.
- Přejde 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 odpověď,
 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
 - ≈ 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?
 - Může být komplikované → standardizace AMQP
 - Př. RabbitMQ
- Message broker
 - Centrální místo pro výměnu zpráv
 - Obecné zprávy (procesy více typů)
 - Rozumí typu zpráv
 - Umí zprávy konvertovat, kategorizovat, ...
 - Lze přidat nové typy (plugin)
 - Systém topiců (publish-subscribe model)
 - Př. Apache Kafka

Inter-Process Communication

- Komunikace mezi procesy na stejném stroji
- Lze uvažovat vše zmíněné
 - Sdílená paměť
 - Mutexy, semaforey, ...
 - 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 → přetížené spoje
- Více typů
 - Flooding (broadcast)
 - uzel preposílá 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

Příště: Čas v DS

Čas a koordinace DS

- Často koordinace na základě času (typicky timestamp).
- Absence globálních hodin
- UTC
 - radiostanice,
 - úrovně,
 - přesnost
- Různý čas na různých uzlech (technologie, mimoběžnost, drift)
- Skutečný čas vs logický čas.

Skutečný čas

Základy

- synchronizace = omezení rozdílu
- nelze synchronizovat plně
- Naivní algoritmus (centrální autorita)
 - K klient požaduje čas u serveru S s UTC
 - K požádá o čas S
 - S odpoví svým aktuálním časem t_s
 - K nastaví čas dle t_s
 - Nefunguje. Proč?
- Omezení: čas nikdy nevracíme zpět (proč?)

Changelog