SNUS - skripta za usmeni

DISTRIBUIRANI SISTEMI

1. Distribuirani računarski sistem (definicija, primeri primene).

Distribuirani računarski sistem je sistem koji je sastavljen od dva ili više računara koji komuniciraju preko mreže, a koje krajnji korisnik, zahvaljujući softveru, doživljava kao jedan skladan sistem. DS je u suprotnosti sa centralizovanim sistemom. Korisnik ne sme da bude svestan kompleksnosti samog sistema, vec treba da ga doživljava kao jedinstvenu celinu.
Razlozi pojave:

- Brz razvoj računara (hardvera)
- Brz razvoj računarskih mreža - glavni razlog
- Složene aplikacije zbog zahteva korisnika
- Neophodnost distribuirane obrade podataka.

Primeri primene:

- Sistem mobilne telefonije
- WWW
- SCADA sistemi
- GPS

2. Osobine distribuiranog sistema.

Prednosti:

- Sklanja raznolikost komponenti od krajnjeg korisnika
- Sklanja način rada (kao ekapsulacija)
- Lako proširiv
- Podržava heterogene komponente
- Uvodi middleware koji nudi interfejs za korišćenje a sakriva kompleksnost sistema od korisnika

Mane:

- Složeniji - potreba za sihronizacijom izmedju računara unutar sistema
- Umanjuje performanse zadataka koji se mogu izvršiti na jednom računaru (zbog cene komunikacije) - kod manjih sistema
- Smanjena sigurnost sistema - većina napada se dešava na mreži, teže ju je zaštiti
- Nije moguće napraviti sistem koji rešava sve teorijske probleme

3. Ciljevi distribuiranog sistema: povezivanje, transparencija, otvorenost, skalabilnost.

- **Povezivanje korisnika i udaljenih resursa**

U DS dolazi do povezivanje korisnika sa resursima, deljenja uredjaja (web laboratorije, štampači, skeneri), deljenja datoteka/skladišta (resursi za čuvanje podataka – dokumenti, email…), rada na daljinu (telekonferencije, e-trgovina). Bezbednost sistema veoma bitna!

- **Transparentnost**

Od krajnjeg korisnika se sakrivaju informacije koje njemu nisu značajne - ne opterećuje se. Za korisnika nije važno gde se i kako ivršava aplikacija. Za njega je važno da sistem doživljava kao jednu celinu.

- **Otvorenost**

Mogućnost dodavanja novih funkcionalnosti bez uticaja na postojeće - funkcionalno proširivanje. Servise obično specificiramo putem interfejsa, gde samo definišemo funkcionalnost, sve metode su virtualne. Interfejsi su suština komponentnog programiranja, jer omogućuju da kompleksan sistem delimo u komponente. Dobro definisan interfejs je kompletan (specificirano je sve što treba implementirati), neutralan (detalji vezani za implementaciju nisu vidljivi spolja), interoperabilan (delovi sistema raznih proizvodjača mogu da rade zajedno i da komuniciraju putem interfejsa), portabilan.

- **Skalabilnost**

Mogućnost horizontalnog i vertikalnog proširivanja sistema. Kako sistem raste, tako i opterećenje sistema raste, pa je potrebno podržati mogućnost naknadnog proširivanja sistema. Skalabilan sistem je u suprotnosti sa centralizovanim: cetralizovan servis, centralizovani podaci, centralizovani algoritam. Decentralizovani algoritmi nemaju kompletnu sliku o stanju sistema, već mora da radi sa podacima koji su mu dostupni i da na osnovu njih uradi najbolji mogući posao.

4. Tipovi distribuiranih sistema (klaster, grid computing).

Postoje dva tipa distribuiranih sistema:

- **Klasteri**
  - Svi računari su istog ili sličnog kapaciteta, isti OS, jedan ili nekoliko vlasnika
  - Okruženje je predvidivo, mreža je predvidiva
  - Pod centralizovanom kontrolom
  - Primeri: Elektrodistribucija, SCADA sistemi
- **Grid computing**
  - Različiti računari, veliki broj OS-a, mnoštvo vlasnika, različito vreme rada
  - Nema centralizovane kontrole
  - Geografski distribuirani
  - Primer: Cloud

5. Povezivanje aplikacija (procesiranje distribuiranih transakcija, globalna integracija aplikacija).

Transakcije su tipično prisutne u aplikacijama koje rade sa bazama podataka. Scenario: mrežna aplikacija se sastoji od jednog ili više servera, a udaljeni korisnici šalju zahteve serveru/serverima, nekoliko zahteva se objedinjuje i izvršava kao distribuirana transakcija. Osobine transakcija:

- **Atomičnost** - izvršavaju se kao celina, nedeljive prema spoljašnjem svetu
- **Konzistentnost** - transakcije ne narušavaju ispravnost podataka
- **Izolovanost** - istovremene transakcije ne utiču jedna na drugu
- **Postojanost** - kada se uspešno završi, promene su trajne

Distribuirane transakcije omogućavaju postojanje podtransakcija (ugnježdjenih transakcija) koje se mogu paralelno izvršavati. Pri neuspehu jedne od podtransakcija, sve ostale se poništavaju. Takodje postoji komponenta koja je zadužena za kontrolu i monitoring izvršavanja transakcije (monitor).

Savremeni koncept za integraciju aplikacija je Service oriented Architecture (SOA) koji podrazumeva da su servisi i korisnici slabo povezani (serverske aplikacije pružaju usluge, klijenti koriste usluge) - koristi se komunikacija zasnovana na porukama. U povezivanju aplikacija koristi se middleware koji je zadužen za prenos poruka.

ARHITEKTURA

6. Softverska i sistemska arhitektura.

Softverska arhitektura predstavlja logičku organizaciju softverskih komponenti. Delimo posao u DS po celinama, tj. komponentama. Komponenta je modularna softverska jedinica, sa jasno definisanim interfejsom. Implementacija samih komponenti je promenljiva. Komponente su gradivne jedinice DS. Softverska arhitektura govori o tome koje komponente postoje i kako se one povezuju i razmenjuju zadatke.

Sistemska arhitektura predstavlja fizičku realizaciju softverskih komponenti, govori o fizičkom rasporedu komponenti. Ona je naredni nivo – određuje tačan broj računara nad kojima će se softver izvršavati, IP adrese… Prelazak iz uopštenog na konkretno rešenje. Komponente se rasporedjuju na stvarne sisteme. Bavi se time gde će se koje komponente nalaziti i kako će komunicirati sa drugim komponentama, konkretne baze, konkretni korisnici. Sistemska arhitektura predstavlja viši nivo apstrakcije; onaj ko implementira sistemsku arhitekturu mora imati najviši nivo znanja i poznavati softver, hardver, mreže..

7. Stilovi softverske arhitekture (slojevita, objektno-orijentisana, ...).

Postoje 4 stila softverske arhitekture:

- **Slojevita arhitektura**
  Najčešće korišćena arhitektura. Komponente koje rade iste ili slične stvari grupišu se u slojeve. Komponente mogu da komuniciraju sa svim komponentama u svom sloju i susedni slojevi mogu medjusobno da komuniciraju. Danas su najčešće troslojne arhitekture: baza, business layer i user interface. Prednosti: jednostavnost, lakoća održavanja, jasno definisane komponente. Mane: više kopija istog podatka, što dovodi do problema konzistencije, što otežava skalabilnost.
- **Objektno-orijentisana arhitektura**
  Slobodnija organizacija komponenti u odnosu na slojevitu arhitekturu. Svaki objekat odgovara komponenti, a jedan objekat ima jednu i samo jednu instancu u čitavom sistemu. Ako neko želi da dobije još jednu instancu objekta, on dobija samo

referencu na postojeći objekat. Prednosti: nema problema konzistentnosti, svako može da komunicira sa svakim - inicijalno bolje performanse. Mana: teško za implementirati na velikim sistemima, slabije performanse zbog uskih grla. Ova arhitektura je dobra za manje sisteme.

- **Podacima uredjena arhitektura**
  Komponente (procesi) koje komuniciraju sa bazom su najvažnije, ostale su manje važne. Stanje u bazi je konzistentno i ukoliko dodje do neuskladjenosti podataka, podaci iz baze se uzimaju kao ispravni. Prednosti: konzistentan sistem. Mane: baza (podaci) je kritični resurs - ukoliko ona otkaže ceo sistem pada, baza se može samo vertikalno skalirati.
- **Arhitektura zasnovana na događajima**
  Kod ovih sistema najvažniji su dogadjaji. Komponente komuniciraju preko propagacije dogadjaja, pri čemu se prenose i podaci vezani za dogadjaj. Ovakvi DS se nazivaju još i Publish/Subscribe sistemi (neke komponente emituju poruke, a druge ih primaju). Ne posmatramo podatke, jer njih ima jako mnogo, već posmatramo samo dogadjaje i na njih se pretplaćujemo. Svaka komponenta reaguje na odredjene dogadjaje. Da bismo postigli brzinu koja je važna za dogadjaje, sami ti dogadjaji tj. podaci vezani za njih moraju biti male veličine. Ovako dobijamo mali broj kvalitetnih informacija iz gomile podataka. Mala poruka se prenosi, ne cela informacija. (Klasičan observer)

U modernim sistemima se često koristi više stilova.

8. Centralizovana arhitektura. Klijent-server model.

Učesnici su dve komponente – server koji implementira servis, i klijent koji koristi uslugu servisa. Tačno se zna ko je klijent a ko server sistema. Server može da bude i klijent. Najčešće ima jedan server I više klijenata. Komunikacija najčešće funkcioniše po principu upit-odgovor. Klijent šalje upit serveru koji traje neko vreme, server obradjuje upit, priprema i šalje odgovor. Klijent preuzima inicijativu slanjem zahteva (započinje komunikaciju). Sve komponente se nalaze na jednom računaru. Mane: slaba konzistentnost, ako otkaže server pada ceo sistem.

9. Višeslojna arhitektura. Middleware.

Često se koristi troslojna arhitektura softvera:

- Sloj korisničkog intrerfejsa (User interface) – unos i prikaz podataka
- Sloj obrade (processing) – obrada zahteva ili rezultata
- Sloj podataka (data) – skladištenje podataka

Komunikacija izmedju slojeva se odvija po Klijent-Server modelu.
Middleware je sloj izmedju distribuiranih aplikacija i platformi (operativnih sistema). Ovaj sistem omogucuje lakše postizanje distributivne transparentnosti. Upotrebu middleware-a nameće arhitektonski stil. Danas se traži prilagodljivi middleware.

DISTRIBUIRANI UPRAVLJAČKI SISTEMI

10. Pogodnosti i primena distribuiranih upravljačkih sistema.

Pogodnosti DRUS-a

Sa stanovišta procesa, primenjuje se zbog povećanja:

- produktivnosti proizvodnje
- kvaliteta proizvoda
- sigurnosti i raspoloživosti postrojenja
- fleksibilnosti rada postrojenja
- kvaliteta uvida u rad postrojenja

Sa stanovišta računarstva I automatike, povećanje automatike kod postrojenja i procesa se stvara primjenom novih:

- računarskih i komunikacijskih tehnologija
- naprednih metoda automatskog upravljanja

Primene DRUS-a

- postrojenja procesne industrije hemijska i petrohemijska postrojenja, rafinerije nafte, željezare i čeličane, cementare, fabrike papira, prehrambena industrija, vodosnabdevanje, postrojenja za tretman otpadnih voda, naftna i gasna polja, …
- proizvodna postrojenja
- energetska postrojenja
- automatika složenijih laboratorijskih postrojenja

11. Hijerarhijska arhitektura distribuiranih upravljačkih sistema.

Opšte je prihvaćena i omogućava funkcionalnu dekompoziciju na nekoliko nivoa

- **Nivo 4**: vođenje preduzeća (najviši nivo složenih industrijskih postrojenja obuhvaćeni inženjerski, ekonomski, komercijalni i kadrovski aspekti vođenja preduzeća)
- **Nivo 3**: vođenje proizvodnje (određivanje redosleda proizvodnje (production scheduling) za jedinice proizvodnog postrojenja u zaisnosti od resursa, nadzor celokupnog proizvodnog postrojenja)
- **Nivo 2**: vođenje postrojenja/procesa (određivanje optimalnih radnih uslova procesa, nadzor, skladištenje i izveštavanje, adaptivno upravljanje (optimalno upravljanje procesom))
- **Nivo 1**: lokalno upravljanje i regulacija (akvizicija procesnih veličina, nadzor procesa/postrojenja i provera ispravnosti sistema, sekvencijalno upravljanje i upravljanje u zatvorenoj petlji)
- **Nivo 0**: tehnički proces (nivo signala i uređaja)

12. Fizička i funkcionalna arhitektura sistema.

Fizička arhitektura je sabirnički orijentisana. Često se funkcije dva ili više funkcionalnih nivoa implementiraju u jednom fizičkom nivou - dobija se jednostavniji DS.
Tipično tri komunikacione mreže:

- **Fieldbus** - povezivanje upravljačkih uređaja sa “pametnim” senzorima i izvršnim organima
- **Upravljačka mreža na nivou postrojenja** - povezivanje više upravljačkih uređaja (i SCADA sistema)
- **Poslovna magistrala** - povezivanje upravljačkog sistema sa ostalim aplikacijama

Funkcionalna arhitektura sistema se odnosi na organizaciju i raspodelu funkcija sistema na različite funkcionalne nivoe ili slojeve. Svaki funkcionalni nivo je odgovoran za specifične zadatke ili funkcije sistema. Ova podela funkcija omogućava modularnost, fleksibilnost i efikasnost u razvoju i održavanju sistema. Tipično, funkcionalna arhitektura sistema ima nekoliko slojeva, a svaki sloj obavlja određene funkcije (npr. nivo upravljanja procesima, uređajima, interfejs korisnika, poslovna logika i aplikacija)

PROCESI

13. Procesi i niti u distribuiranom sistemu.

Procesi

Proces predstavlja izvršavanje određenog programa ili aplikacije. U distribuiranom sistemu, procesi su samostalne jedinice izvršavanja koje mogu se izvoditi na različitim računarima ili čvorovima u mreži. Svaki proces ima svoj sopstveni prostor za memoriju, registre i izvršne kontekste. To omogućava nezavisno izvršavanje i održavanje stanja između procesa.

U nadzorno upravljačkom softveru, procesi se često koriste za nadzor i upravljanje industrijskim procesima. Na primer, procesi se mogu koristiti za prikupljanje podataka sa senzora, obradu tih podataka i upravljanje različitim aspektima industrijskih sistema, kao što su motori, ventili, osvetljenje itd. Svaki proces može imati svoje definisane zadatke i ciljeve koje treba izvršiti.

Niti

Niti su manje jedinice izvršavanja unutar procesa. Niti dele zajednički prostor za memoriju sa ostalim nitima u istom procesu. Kada se koriste niti, moguće je paralelno izvršavanje različitih delova programa unutar istog procesa.

U distribuiranim sistemima, niti su korisne za paralelizaciju zadatka na različitim čvorovima ili za obradu velikog broja zahteva istovremeno. U kontekstu SCADA softvera, niti se često koriste za simultano upravljanje više različitih procesa ili za obradu podataka u realnom vremenu.

Na primer, u industrijskom okruženju, SCADA softver može imati niti za nadzor različitih senzora istovremeno. Svaka nit bi se bavila praćenjem i obradom podataka sa određenog senzora. Paralelno izvršavanje tih niti omogućava brzo i efikasno praćenje stanja različitih senzora i pravovremenu reakciju u slučaju problema.

Prednosti procesa i niti u distribuiranom sistemu:

- Paralelizacija: Kombinovanje procesa i niti omogućava paralelno izvršavanje različitih zadataka ili delova programa. To dovodi do bolje iskorišćenosti resursa i povećanja ukupnih performansi sistema.

- Otpornost na greške: Distribuirani sistemi mogu imati više kopija istih procesa ili niti koje se izvršavaju na različitim čvorovima. U slučaju da jedan čvor ili nit doživi neuspeh, ostale kopije mogu nastaviti sa radom. Ovo povećava otpornost sistema na greške i osigurava kontinuirano funkcionisanje.
- Efikasno korišćenje resursa: Distribuirani sistemi mogu rasporediti procese i niti na različite čvorove u mreži. Na taj način se efikasno koriste raspoloživi procesorski resursi i memorija. Ova distribucija omogućava bolje iskorišćenje resursa i veću skalabilnost sistema.

14. Niti i dizajn servera.

Niti omogućavaju paralelno izvršavanje različitih zadataka u okviru servera. U distribuiranim sistemima, serveri mogu biti raspoređeni na različitim čvorovima mreže kako bi se omogućilo efikasno upravljanje i obrada velikog broja zahteva.
Korišćenje niti u dizajnu servera omogućava simultano upravljanje više zahteva od strane klijenata. Na primer, u SCADA sistemu, klijenti mogu slati zahteve za praćenje, upravljanje ili prikupljanje podataka. Niti omogućavaju serveru da istovremeno obradi ove zahteve, omogućavajući brzo i efikasno odgovaranje na različite zahteve.
Serveri u distribuiranim SCADA sistemima često koriste niti za praćenje i obradu podataka u realnom vremenu. Niti se mogu koristiti za kontinuirano praćenje senzora i drugih ulaznih podataka, i pravovremeno reagovanje na promene stanja ili alarme.

SINHRONIZACIJA

15. Utvrđivanje redosleda događanja. Sinhronizacija logičkog vremena.

Pored komunikacije procesa, u DS je veoma bitno kako oni saradjuju i kako se sinhronišu. Neophodno je odrediti kontrolisan pristup deljenim resursima i utvrditi redosled dogadjaja. Sinhronizacija procesa unutar DS je mnogo složenija od sinhronizacije procesa u jednom mašini. Potrebno je: utvrditi redosled dogadjaja, obezbediti isključivi pristup deljenim resursima, sinhronizovati rad procesa.
Utvrdjivanje redosleda dogadjanja - odnosi se na sinhronizaciju rada distribuiranih procesa. Problem: nema globalnog sata, tj. procesi na razlicitim masšnama imaju sopstvene predstave o vremenu.
Sinhronizacija logičkog vremena - u svim real-time sistemima su neophodni časovnici (merači stvarnog vremena). Postoji više načina za njihovu sinhronizaciju: jedan server koji svima šalje vreme ili server šalje klijentima vreme, prima razlile i koriguje se ili upotreba GSP-a.

Osnovna ideja kod sinhronizacije vremena jeste da se razmenjuju informacije o vrednostima sata, pri čemu se vodi računa o vremenu neophodnom za slanje i primanje poruka. Tačnost algoritma zavisi od kašnjenja u komunikaciji. Međutim, samo poznavanje tačnog vremena

nije toliko važno u komunikaciji koliko je važan redosled izvrđenja pojedinih dogadjaja (Lamportov logicki časovnik)
Lamportov logicki časovnik: svakom dogadjaju E se dodeljuje logička vremenska značka (timestamp), informacije o vremenu se prenose porukama gde slanje poruke predhodi njenom prijemu (ako dogadjaj A predhodi B tada je c(A) < c(B)).

16. Deljenje resursa i međusobna isključivost. Algoritmi za dodelu pristupa.

Medjusobna isključivost u pristupu procesa treba da spreči korumpiranje resursa (da bi njegovi podaci ostali konzistentni). Međusobna isključivost obezbedjuje da u svakom trenutku pristup deljenom resursu ima samo jedan proces. Primenjuju se sinhronizacioni algoritmi.
Postoje brojni algorimi za dozvolu pristupa deljenom resursu (svi algoritmi su osetljivi na greške u komunikaciji i otkaze u procesima):

- **Algoritmi zasnovani na dozvoli pristupa** (centralizovana, decentralizovana i distribuirana odluka)
- **Algorimi zasnovani na prosledjivanju tokena**

Centralizovana odluka: Jedan proces se proglasi za koordinatora, ostali procesi od njega traže pristup deljenom resursu. Kordinator odlučuje o redosledu pristupa.
Problem: kordinator je usko grlo.
Decentralizovana odluka: Kordinatori su umnoženi n puta. Kada proces poželi da pristupi resursu treba da dobije većinu glasova kordinatora tj. m>n/2.
Prednost: neosetljiv na gubitak jednog koordinatora.
Problem: kordinator može da zaboravi kako je glasao; veliki broj poruka za razmenu.
Distribuirana odluka: Proces šalje poruku svim ostalim procesima da želi da pristupi deljenom resursu (prosledjuje i logičko vreme zahteva), kada mu svi procesi odgovore on pristupa resursu. Pita sve ostale za dozvolu i kad dobije dozvolu od svih onda pristupa.
Problem: ako barem jedan proces zataji, nema odluke.

Algoritam prosledjivanja tokena: U sistemu postoji token koji cirkuliše izmedju procesa, samo vlasnik tokena ima pravo da pristupa resursu. Kada završi posao, proces token prosledjuje drugom procesu.
Prednost: svaki proces dobija šansu da pristupi resursu.
Problem: gubitak tokena je ozbiljan problem

17. Sinhronizacija upotrebom koordinatora. Izbor koordinatora.

Sinhronizacija procesa zahteva postojanje posebnog procesa - koordinatora. On može da bude unapred izabran i nepromenljiv ili dinamički izabran od strane drugih procesa. Dva osnovna algoritma za biranje koordinatora:

- **Bully algoritam** - Preduslov je da svi procesi imaju jedinstven broj. Jedan od procesa P shvati da nema koordinatora i pokrene izbore. On postaje inicijator i šalje poruku "izbori" svim procesima većeg broja tj. prioriteta, ako niko ne odgovori, proces P postaje koodinator, u suprotnom onaj ko je odgovorio postaje inicijator. Na kraju se izabere proces koji ima najveći prioritet.

- **Algoritam prstena** - procesi se nalaze u lancu i imaju brojeve (priotitete). Jedan od procesa P shvati da nema koordinatora i pokrene izbore. Šalje poruku u koju svaki proces ubaci svoj broj. Kada se krug zatvori P odredi koordinatoora sa najvišim prioritetom i obavesti sve procese.

REPLIKACIJA I KONZISTENCIJA PODATAKA

18. Replikacija podataka.

Replikacija podataka je zahtev da iste podatke imamo na više mesta. Potreba postoji zbog: povećanja pouzdanosti sistema (ako oštetimo jednu repliku koristimo podatke iz druge) i zbog poboljšanja performans,e procesu. Zbog replikacije se javlja problem konzistentnosti – jer se promene replike moraju propagirati čime se narušava performansa sistema. Promena podatka se mora propagirati u sve kopije I redosled operacija mora biti očuvan. Ovo je veoma neugodno za performanse i zato se uvodi model konzistentnosti koji umanjuje ograničenja pri replikaciji.

19. Konzistencija podataka. Modeli konzistencije.

Konzistencija podataka se odnosi na operacije čitanja i pisanja nad deljenim podacima u skladištu podataka (deljena memorija, baza podataka ili fajl sistem). Svaki proces koji pristupa podacima ima lokalnu kopiju celog skladišta – repliku. Najteži problem je kada imamo puno kopija, kada svi procesi hoće da pišu i imamo vremensko ograničenje za izvršavanje. Model konzistencije je ugovor procesa i lokalnog skladišta podataka (ako proces poštuje pravila, skladište će biti ispravno). Npr. nakon write operacije read treba da vrati ono sto je upravo promenjeno. Pošto nemamo globalni sat i ne znamo koja je operacija bila poslednja, različiti modeli konzistencije nam daju ograničenja koje podatke mora da vrati read operacija. Modeli konzistencije su:

- **Kontinualna konzistencija** – cilj je postaviti granice na:
  - Maksimalne dozvoljene razlike u numeričkim vrednostima izmedju replika
  - Maksimalno dozvoljeno odlaganje osvežavanja podataka - Definiše se period u kome se smatra da je replika konzistenta iako se tokom tog perioda događaju promene vrednosti
  - Dozvoljene razlike u redosledu operacija
- Konzistentan redosled operacija – je osnova za brojne modele konzistencije kao npr:
  - **Sekvencijalna konzistencija** - bitan je redosled opreacija ali je dozvoljeno kašnjenje (primer transakcije u banci, nije nam bitno da se odmah izvrši ali je bitno kojim redosledom se izvršavaju transakcije).
  - **Uzročna (kauzalna) konzistencija** - obezbeđuje da operacije koje potencijalno zavise jedna od druge moraju biti primenjene prema zahtevanom redosledu. Za operacije koje nisu zavisne redosled primene u replikama nije bitan

- **Model klijent centrične konzistencije** - ovaj model je bitan za mobilne klijente. Pretpostavlja se da se isti klijent povezuje na različite replike. Suština je da podaci ne moraju biti smešteni na istom mestu, najbitnije je da klijent uvek zatekne stanje koje je poslednje ostavio. Ne razmatra se slučaj da se podaci dele između više klijenata, nego se razmatra konzistentnost podataka prema samo jednom klijentu. (Google search - svako može da dobija različite stvari za isti search)

20. Protokoli distribucije promena u replike

Protokoli distribucije promena nam govori ko, gde i kada vrsi replikaciju. Vrste replikacija:

- **Stalne (permanentne) replike** – početni skup replika koje čine distribuirano skladište podataka, ima ih malo, 100% su poravnate i koriste se za baze podataka.
- **Server-inicirane replike** - Npr. ako dodje do zagušenja onda se pravi kopija tamo odakle dolazi najvise zahteva. Kopije se prave dinamički na inicijativu servera i on je odgovoran. Koriste se kada je cilj povecanja performansi.
- **Klijent-inicirane replike** – kopije se prave dinamički na inicijativu klijenta i on je odgovoran. Nazivaju se jos i keš (client cache) – lokalna kopija privremeno čuva skorašnje očitane podatke. Ovim se ubrzava pristup podacima. Udaljena skladišta podataka (SP) ne vode brigu o kešu klijenta – Klijent ga sam održava – Klijent očitava podatke iz SP i smešta ih u keš. Problem je kada dodje do promene podatka. Pogodno je da isti keš deli više klijenata i tada se povećava broj „pogodaka“ pa se traženi podaci nalaze u kešu.

Ako dodje do promene podatka potrebno je izvršiti propagaciju, šta se propagira? Mogucnosti propagacije podataka su:

- Propagira se obaveštenje da je nastala promena – navodi se koji deo podataka je promenjen. Fajl koji je promenjen se obeleži da je promenjen i kada mu se pristupa mora se prvo osvežiti. Ovakav nacin može da optereti mrežu zbog velikog broja dojava ali radi dobro kada imamo dosta promena - Read/Write je malo.
- Propagacija podataka – korisno kada je Read/Write veliko. Mogu se beležiti samo promene i one slati u kopije, pogodno pakovanje više promena u jednu poruku.
- Prenose se operacije (koje prave izmene) sa kopije na kopiju – Slanje operacija koje će u kopijama izazvati istu promenu podataka. Ako su operacije složene, jednostavnije je preneti rezultate.

TOLERANCIJA NA OTKAZE

21. Sistem tolerantan na otkaze (osobina, cilj).

Osobina distribuiranog sistema je da postoji mogućnost delimičnog otkaza. Delimični otkaz znači da jedan deo sistema ne radi, tj. neka komponenta je otkazala ali klijent i dalje ima sve funkcionalnosti samo sa smanjenim performansama. Može uticati loše da rad ostatka sistema a za neke delove je potpuno nebitan.

Cilj je dizajnirati sistem tako da se automatski oporavlja od delimičnog otkaza tj. da pri delimičnom otkazu sistem može da nastavi sa radom na prihvatljiv način. Tako dizajniran sistem je otporan na otkaze i zove se fault tolerant.

22. Osobine distribuiranog sistema: raspoloživost, pouzdanost, sigurnost, održavanost.

Osobine koje pokazuju koliko je sistem dobro pouzdan su:

- **Raspoloživost** – osobina sistema da radi bez greske; procenat kolika je verovatnoća da radi u svakom trenutku. Odnosi se na TRENUTAK
- **Pouzdanost** - osobina da sistem radi kontinualno bez ijedne greške (period između dve greške). Odnosi se na PERIOD
- **Sigurnost** – privremeni otkaz ne uzrokuje katastrofu, osobina da ako I dodje do kvara da ne dodje do katastrofe, najbitnije je spasiti ljudske živote, zatim spašavanje opreme, sigurnost se poboljšava dodavanjem redudanse i posebne opreme
- **Održivost** – osobina koliko brzo sistem može da se oporavi od greške, kada dodje do prekida sistema, koliko mu treba vremena da se oporavi. (Izgoreli su hard diskovi, da li će mi trebati 2 sata ili dva meseca da ih popravim)

23. Uzroci grešaka i tipovi otkaza. Problemi klijent-server komunikacije.

Uzroci grešaka:

- **Prolazn**i (ne znamo koliko će trajati - nestanak interneta, ako ne dobijemo odgovor servisa, probati par puta)
- **Povremeni** (opasni jer ne znamo uzrok i ne znamo kad će se ponovo desiti, kad se otkrije uzrok, najlakši za rešavanje)
- **Trajni** (sistem je otkazao u potpunosti, najlakči za otkrivanje, manje opasno nego povremeni).

Tipovi otkaza:

- **Crash failure** - server se blokira, ali radi korektno dok se ne blokira
- **Ommission failure** - server otkazuje kod odgovora na zahtev klijenta. Specijalni slučajevi - **receive** (otkaz kod prijema poruke) i **send** (otkaz kod slanja poruke)
- **Timing failure** - odgovor servera kasni više od specificiranog perioda
- **Response failure** - odgovor servera je nekorektan. Specijalni slučajevi **value** (vrednost u odgovoru je loša) i **state transition** (server odstupa od koreknog toka aktivnosti)
- **Arbitrary failure (Vizantijski otkaz)** - server može dati proizvoljan odgovor u proizvoljno vreme

24. Skrivanje otkaza upotrebom redundanse. Grupa procesa. Atomski multikasting.

Jedini način i ključna tehnika da se reši otkazivanje sistema je suvišnost (redudansa).

- **Suvišnost informacija** - dodatni biti, npr. Hamingov kod, kontrolna suma
- **Suvišnost u vremenu** - akcija se ponavlja, npr. neuspela komunikacija

- **Fizička suvišnost** - u hardveru i/ili softveru (2 bubrega, 2 plućna krila, 747 ima 4 motora, 4 fudbalskih sudija)

Atomski multikasting je tehnika koja se koristi u računarstvu kako bi se obezbedila pouzdana i sigurna komunikacija između više uređaja ili čvorova u mreži. U kontekstu rešavanja otkazivanja sistema, atomski multikasting se može primeniti na nivou grupnih procesa.
Grupa procesa se sastoji od više procesa koji rade zajedno kako bi obavili određeni zadatak. Atomski multikasting omogućava grupi procesa da sinhronizovano izvršava određene akcije. To znači da se poruke ili zahtevi koje šalje grupa procesa šalju na način da su atomski, odnosno da se ili svi procesi u grupi uspešno obaveste o poruci ili nijedan.
Primenjujući atomski multikasting u kontekstu suvišnosti (redudanse), grupa procesa može koristiti ovu tehniku kako bi osigurala da se akcije koje se ponavljaju, kao što je slanje informacija ili zahteva, izvršavaju na pouzdan način. Na primer, ako se otkrije da jedan od procesa u grupi neuspešno šalje informacije ili zahteve, ostali procesi mogu preuzeti tu ulogu i obezbediti da poruke budu ispravno dostavljene svim članovima grupe. Ovom kombinacijom suvišnosti (redudanse) i atomskog multikastinga, sistem se može zaštititi od otkazivanja i obezbediti da se važne akcije ili poruke uspešno izvršavaju čak i u slučaju neuspeha pojedinih komponenti ili procesa.

25. Distribuirani Commit protokol. Dvofazni i trofazni Commit protokol.

Protokol koji koristimo kada je potrebno da procesi iz svoje grupe razmene poruke i da se poruka primeni ili u svakom procesu ili nigde. Rešava se uz pomoć koordinatora i učesnika.
Dvofazni commit protokol – ima dve faze:

- **Pripremna faza** – Glavni kordinator obaveštava ostale procese da će se desiti akcija i pita ih da li su spremni da odrade svoj deo posla.
- **Commit faza** – Ako su svi javili da su spremni transakcija se izvrši. Ako nisu svi spremni, šalje se Abort, transkacija se prekida.

Trofazni Commit protokol je proširena verzija dvofaznog Commit protokola koja omogućava rešavanje problema koji se javljaju kada koordinator doživi neuspeh ili pad. Trofazni protokol dodaje dodatnu fazu kako bi se osiguralo da se transakcija uspešno izvrši čak i u slučaju pada koordinatora. Prve dve faze su iste kao kod dvofaznog protokola, a dodatna je treća faza:

- **Faza potvrde** - Koordinator šalje poruku potvrde (acknowledgment) svim učesnicima kako bi ih obavestio da je transakcija uspešno izvršena. Učesnici tada potvrđuju prijem ove poruke. Ova faza je posebno važna za rešavanje problema pada koordinatora. Ako koordinator ne uspe da primi potvrde od učesnika, onda će, kada se oporavi, pretpostaviti da se transakcija nije uspešno izvršila i ponovno će je pokrenuti.

26. Oporavak od otkaza (Checkpoint, beleženje poruka, postojano skladište podataka).

Nakon otkaza proces treba da se oporavi i dovede u korektno stanje. Rekonstrukciju stanja omogućavaju: Checkpoint-i i Beleženje poruka
Checkpoint (oporavak u nazad) je da se stanje sistema povremeno beleži. Npr na svaku promenu se kopira čitava baza podataka. Problem je što to moze da traje i možda ponovo nastane greška.
Beleženje poruka (oporavak u napred) je da se beleže samo dogadjaji.

Najbolje rešenje jeste kombinovanje ova dva načina tako sto se checkpoint redje pravi, pošiljaoc beleži poruku pre nego je pošalje, primaoc beleži poruku pre nego je prosledi aplikaciji. Oporavak ide tako što se učita stanje najskorijeg checkpoint-a i primene se poruke nastale posle snimanja stanja.
Važno nam je i da imamo neki dodatni bafer koji će čuvati dogadjaje koji su se desili dok je baza podataka bila u otkazu. Kada se baza oporavi zapisuju se poruke koje su u medjuvremenu sacuvane u ovom baferu.

PRAKTIČNA PREDAVANJA

27. Profibus, ModBus, FieldBus
28. Smart Alarming
29. SCADA u transportnoj industriji (drumski, zeljeznicki, vodeni, vazdusni)
30. SCADA u Distirbucijama (struje, vode, gasa)
31. SCADA u procesnoj industriji
32. SCADA u biomedicinskom inzenjeringu
33. Real-time video menadžment za SCADA

Iz Bojanove skripta:

SCADA

- **Definicija I arhitektura klasičnog SCADA sistema**

SCADA (Supervisory Control And Data Acquisition) sistem je sistem namjenjen za optimizaciju proizvodnje.
Najvaznija karakteristika SCADE je u tome sto je to real time sistem koji blagovremeno i pouzdano tacno dobavlja podatke.
SCADA se sastoji od master stanice (koja prikazuje prikupljene podatke i preko koje operater izvršava udaljeno komandovanje) i veceg broja RTU (koji prikupljaju podatke iz polja i povezani su komunikacionim linijama sa master stanicom, najčešće wireless tehnologijom).
Najznacajnije mesto unutar SCADA hijerarhije predstavlja kontolna soba. Greske koje se dese na ovom nivou su najskuplje, pa se shodno tome sav hardver, softver, komunikacione mreze i oprema u ovoj sobi uvek poduplavaju.

- **Arhitektura RTU i komunikacija sa RTU**

RTU po svojoj strukturi predstavlja standardni racunarski sistem sa posebnim modulima za ulaz/izlaz. RTU se sastoji od klasicnih elemenata svakog racunarskog sistema kao sto su: napajanje, procesor, radna i stalna memorija, magistrala, modem, ali pored toga RTU sadrzi i neke dodatne komponente kao sto su ulazno/izlazni moduli tj. analogne i digitalne ulaze i izlaze. Ovi moduli sluze za povezivanje racunarskih sistema sa mernim instrumentima, rade konverziju i vode racuna o velicini signala kako ne bi bio prejak da sprzi plocu.

Signal unutar RTU sistema je najcesce izmedju 4 i 20 mA, odnosno nikad mu donja granica nije 0. Razlog tome je sto ukoliko nam na ulaz dodje 0, sistem nece moci da razlikuje da li je to dolazak niskog napona (tj.logicke nule) ili je doslo do kvara na mernim aparatima. RTU se fizicki bolje izoluje i pravi se od cvrstih materijala kako bi bolje izdrzali uslove rada.

- Ciklicno prozivanje – centrala inicira komunikaciju. Master centrala redom, tj. u krug (round robin) proziva sve svoje merace tj. RTU i proverava da li je doslo do promene njihovog stanje. Prednosti: jednostavna implementacija.
Mane: potencijalno kasnjenje i puno nepotrebne komunikacije.
- Dojave – RTU iniciraju komunikaciju. RTU prikuplja podatke i javlja se centrali kada se desi promena unutar njega.
Prednosti: nema nepotrebne komunikacije i kasnjenja
Mane: tesko za implementaciju (RTU-ovi moraju biti pametniji) i moguca kolizija (ukoliko dodje do nekog veceg problema u sistemu, svi meraci ce hteti da taj problem dojave centrali)

Najbolji parametar za izbor komunikacije je broj uredjaja u sistemu i cena. Ukoliko imamo manje uredjaja bolje je ciklicno prozivanje, u suprotom slucaju treba koristiti dojave.
Ukoliko zelimo da utvrdimo da li neki od uredjaja unutar sistema salje pogresne podatke, jednostavno stavimo da 3 uredjaja (ili vise u zavisnosti od prioriteta onog sto merimo) da

vrse merenja nad istim podacima. Ukoliko nam 2 od 3 uredjaja salju iste podatke, znamo da je onaj 3. neispravan.

- **Osnovni zadaci (posistemi) SCADA Sistema**

- Podsistem za definisanje veličina u kome se definišu veličine i njihove osobine kao što su gornja i donja granica vrednosti veličine, jedinica mere, vreme očitavanja, inženjerska jedinica itd. Ulazne veličine predstavljaju vrednosti izmerenih fizičkih veličina iz procesa, a izlazne veličine vrednosti koje se šalju ka upravljačkim uređajima. Često se mogu definisati i memorijske veličine (koje služe za proračune) i sistemske veličine koje su specifične za upotrebljeni program.

- Podsistem za alarme koji služi za definisanje i prikaz alarmnih stanja u sitemu. Alarmna stanja mogu prestavljati nedozvoljenu ili kritičnu vrednost veličine kao i nedozvoljenu akciju ili komandu operatera. Svaki alarm ima svoje osobine kao što su nivo ozbiljnosti alarma, mesto nastanka, kategorija, poruka koja se vezuje za alarm i slično. Podsistem za alarme omogućuje promenu stanja alarma putem operacije potvrde i brisanja.

- Podsistem za prikaz trendova u kome se prikazuju poslednje promene vrednosti veličina (trendovi u realnom vremenu) i istorijat promene vrednosti veličina u toku dužeg vremenskog . Periodi (histogrami) - Dobro osmišljeni podsistemi za prikaz trendova omogućuju i uporedni prikaz više veličina kao i arhiviranje dijagrama.

- Podsistem za recepture omogućuje zadavanje više vrednosti veličina kao željene promene vrednosti veličina u vremenu. Koristi se kada unutar sistema postoje prelazi izmedju stanja kada je potrebno menjati veliki broj velicina u tom sistemu, da ne bismo menjali rucno velicine svaki put, uvodimo neka predefinisanja stanja u kojima se sistem nalazi pri promenama izmedju stanja sistema.

- Podsistem za izveštaje u kome se formiraju izveštaji o promenama vrednosti veličina, alarmima, akcijama operatera i ostalim aspektima rada postrojenja.

- Grafički podsistem prikazuje stanje postrojenja u obliku koji je najpregledniji za čoveka (operatera) kako bi on mogao pravovremeno odreagovati na promenu stanja sistema. Osnovna ideja je da se letimičnim pogledom na ekran uoče nepravilnosti u radu postrojenja, da bi se brzo reagovalo i sprečilo neželjeno ponašanje. Vrednosti veličina se najčešće prikazuju u obliku brojeva ili “dinamičkih slika”, čime se olakšava uočavanje promena na slici. Pored prikaza stanja sistema grafički podsistem treba da omogući i izvršavanje neke akcije od strane operatera. Na primer klikom miša na neki objekat može se pokrenuti izvršavanje nekog ranije definisanog makroa ili skripta. U većini dostupnih sistema omogućeno je pisanje makroa u VBA (Visual Basic for Application) programskom jeziku koji se odlikuje jednostavnom sintaksom.

- Komunikacioni podsistem omogućuje povezivanje SCADA sistema sa fizičkim uređajima koji vrše neposredan nadzor i upravljanje (PLC - Programmable Logic Controller - dizajniran hardver za kontrolu industrijski procesa - npr. rad motora). Najčešće se ovo povezivanje vrši preko drajvera koji su razvijeni od strane proizvođača merne i upravljačke opreme. U

SCADA softverskim sistemima nezavisnih proizvođača postoji velika paleta drajvera za opremu različitih proizvođača, dok je kod proizvođača SCADA softvera i upravljačke opreme akcenat stavlja na drajvere za sopstvene uređaje.

- Podsistem za pristup bazama podataka (DBMS - Database Management System) omogućuje trajno čuvanje i pregled podataka u relacionim bazama podataka. Ranija rešenja su beležila podatke u datoteke u nestandardnom obliku. Novija rešenja koriste neki od standardnih načina arhiviranja podataka koji omogućuju korisniku lak pristup podacima kao i pristup podacima iz drugih softverskih sistema. Na Microsoft Windows operativnim sistemima često se koristi ODBC (Open Database Connectivity) i nešto savremenija ADO (ActiveX Data Object) tehnologija. Upotreba ovih tehnologija omogućuje lakšu pretragu podataka kao i formiranje izveštaja pomoću SQL (Structured Query Language) jezika.

- Mrežni podsistem omogućuje povezivanje ostalih podsistema SCADA programa. Brzina rada celog sistema u mnogome zavisi od programskog rešenja ovog sistema. Obično se rešenja oslanjaju na operativni sistem i koriste poznate protokole (transportnog nivoa). Jedna grupa rešenja koristi DDE (Dynamic Data Exchage) tehnologiju koja je jednostavna za korištenje a pokazuje dobre performanse u sistemima u kojima se prenosi mala količina podataka. U složenijim sistemima često je koriste sistemi distribuiranih objekata koji se zanivaju na DCOM - (Distributed Component Object Model) i CORBA - (Common Object Request Broker Architecture) tehnologijama.

OPC - OPEN PLATFORM COMMUNICATION

- **Ciljevi OPC fondacije**

OPC = OLE for Process Control
OLE = Object Linking and Embedding

Mjesta primjene:
- Povezivanje komponenti distribuiranog upravljačkog sistema
- Pristup podacima postojećeg upravljačkog sistema – Napravljen je OPC softverski omotač
- Pristup podacima upravljačkih uređaja (mernoakvizicionih uređaja)

OPC fondacija je nastala 90-tih godina na zahtev korisnika, jer se javila potreba za standardizacijom medjuprocesne komunikacije u industriji.
OPC standard je skup interfejsa, metoda sa svim parametrima i pravila koja moraju biti postovana od strane svih proizvodjaca.
OPC fondacija nije implementirala metode vec svaki proizvodjac, a fondacija samo zadaje parametre.

Dve najznacajnije specifikacije unutar OPC standarda su:
- DA specifikacija (Data Acess) -- PRISTUP VELIČINAMA
- HDA (History Data Acess) -- PRISTUP VELIČINAMA NA ISTORIJSKOM NIVOU

Sve specifikacije su klijent/server orijentisane. Klijent je taj koji inicira samu komunikaciju, server prihvata njegov zahtev, obradjuje ga i prosledjuje odgovor nazad klijentu. Vazno je napomenuti da je komunikacija izmedju klijenta i servera moze biti sinhrona i asinhrona. Sinhrona komunikacija je blokirajuca, sto znaci da klijent ostaje blokiran, sve dok mu server ne odgovori. Prednost sinhrone komunikacije jeste u tome sto smo na taj nacin sigurni da ce klijent uvek dobiti odgovor, dok mu je mana sama blokiranost klijenta.
Asinhrona komunikacija nije blokirajuca, sto je i sama prednost ovog nacina komunikacije, medjutim prilikom ovakvog vida komunikacije klijent nikad ne zna kada ce mu stici odgovor od servera.

Sami OPC standardi su definisani u C++ jeziku, medjutim, postoje mnogi omotaci (wreperi) koji omogucuju i serveru pristupamo i na drugim jezicima.
Brz pristup trenutnim velicinama. Uz vrednost se cuvaju vreme i kvalitet vrednosti.
Unutar OPC fondacije postoje 3 grupe objekata:

- Server - predstavlja najvisi objekat u hijerarhiji
- Item - predstavljaju fizicke velicine kojima se upravlj
- Grupa - nema fizicku reprezentaciju, predstavlja skup item-a, omogucuje optimalan pristup velicinama

OPC Group Object
Glavna metoda unutar OPC Sever Interfejsa jeste add_group

- **OPC Server Veličina (OPC DA)**

Data Acess specifikacija sluzi za brz pristup trenutnim vrednostima velicina. Postoje 3 atibuta koja blize odredjuju samu vrednost velicine:

- Vrednost - npr. temperatura je 44 C
- Timestamp (vreme kad se ocita vrijednost)
- Kvalitet - moze biti NULL, questionable, bad... Sama specifikacija podrzava i sinhronu i asinhronu komunikaciju.
- Dodatni atributi

- **OPC Istorijski server (OPC HDA)**

Ova specifikacija sluzi za cuvanje istorijskih velicina i njihov prikaz, tj. ocitavanje. Pomocu Playback interfejsa omoguceno je da u ponovljenom, realnom vremenu vidimo promene vrednosti podataka, alarme i slicno. Klijent moze da bira na koliko sekundi zeli citanje podataka. Takodje, unutar HDA moguce je specifirati vremenski interval za neku velicinu.

OSNOVE CLOUD SISTEMA

- **Definicija Cloud Computinga**

Cloud je operativni sistem za velike Data Centre. Služi za upravljanje resursima. Definiše životni ciklus aplikacije. Aplikacijama pruža viši nivo abstrakcije pri korišćenju compute,

storage i network resursa. Lako se skalira i omogućuje klijentima iluziju da imaju na raspolaganju beskonačne resurse.

- **Prednosti Cloud Computing-a**

Ekonomija velikih brojeva - Veliki dobavljači (vlasnici Cloud-a) nabljaju servere mnogo jeftinije nego sto mi možemo.
Pay as you go - Ako su vam potrebni dodatni resursi nemorate kupovati nove servere nego ih iznajmite od Cloud provajdera. Ako vam ne trebaju postojeći resusi otkažite njihovu upotrebu i ne morate plaćati za njih.
Niska početna cena - Nemate početna ulaganja i dugoročne ugovore.
Visoka dostupnost - Vaše aplikcije rade 24/7/365.

- **Nivoi Usluga u Cloud-u**

Nivoi usluga:
• SaaS – Software as Service: nudi se poslovna logika, korisnika ne interesuje koji algoritmi se implementiraju, klijent samo placa uslugu.
• IaaS – Infrastructure as Service: dobija se operativni sistem prazan.
• PaaS – Platform as Service: dobija se baza podataka.
• Haas - Hardware as Service: dobija se hardver. Prilikom kupovine odredjenog nivoa prave se ugovori o pruzanju usluga.

- **Windows Azure Arhitektura**

Fabric je mreža međusobno povezanih Node-ova.
Azure Fabric Controller je servis koji nadzire, održavana i u slučaju potrebe startuje nove Node-ove.
Osnovni Servisi u Windows Azure:
• Compute: Izvršava programe na Windows Server 2008.
• Storage: Omogućuje siguran pristup podacima.
• Network: Omogućuje komunikaciju sa spoljnim aplikacijama (Service Bus).

- **Resursi za proračun i Komunikaciju između Rola**

Web role je jedna instance neke aplikacije,a worker role je servis koji sluzi za proracun. Komunikacija izmedju rola se izvrsava preko redova sto je preporucen nacin komunikacije za siguran prenos podataka, pored ovoga komunikacija moze ici I preko tcp ili http komunikacije.

- **Azure Storage (Table, BLOB, Queue, SQL).**

Windows Azure Storage - Omogućuju čuvanje ogromne količine podataka (PB). Lako skaliranje. Visoko pouzdani sistemi za čuvanje.

Podaci su smešteni u velikim farmama servera.
Skalabilnost - Podaci se mogu distirbuirati na veliki broj nodova. Pristup podacima je balansiran.
Pouzdanost - Svi podaci su replicirani u više nodova pri svakom upisu (3 replike se na laze u različitim fault domemima). Kada uređaj zakaže, podaci se repliciraju na novi nod.

Table
Table omogućuju pristup struktuiranim podacima.
Table sadrži skup entiteta. Entiteti sarže skup property-ja. Property sadrži:

- ime
- Definiciju tipa (Int32, Int64, double, string, bool, DateTime, GUID, byte array).

Table nemaju fiksnu šemu, tj. strutktura svakog entiteta može da bude drugačija. Veličina jednog entiteta je ograničena na 1MB.

Blob
Blob Container omogućuje pristup skupu blobova. Način pristupa se postavlja na nivou Container-a: Javni ili privatni. Container mogu da imaju metadata.
Blob čuva velike pakete podataka.
Tipovi Blob-a:

- Block Blob su otimizovani za striming. Maksimalna veličina blob-a : 200 GB. Blob se snimaju blokovima (max. 4 MB).
- Page Blob su optimizovani za slučajan pristup. Maksimalna veličina blob-a : 1TB.

Queue
Queues omogućuje pouzdanu asinhronu razmenu poruka.
Pouzdanost - Poruke će biti ispručene tek kada klijent potvrdi da je preuzeo poruku. Poruke se obrađuju sigurno barem jedan put.
Razdvajanje klijenta i servera - Različiti delovi sistema mogu biti implementirani koristeći različite tehnologije. Klijent i server ne moraju da budu dostupni istovremeno.
Skalabilnost - Queue kompenzuju pikove u opterećenju i kompenzuju otkaze u hardware-u. Ako se uoči povećanje reda čekanja: može se povećati broj računara koji opslužuju sistem.

SQL
SQL Azure je relaciona baza podataka koja je dostupna na Cloud bazirani načina.
SQL Azure u osnovi omogućuje istu funkcionalnost kao i Microsoft SQL Server koji radi lokalno.
SQL Azure omogućuje pristup preko Tabular Data Stream (TDS) endpoint. TDS je mrežni protokol koji koristi i SQL Server.