Horisontellt skalbara molntjänster

Sebastian Lindholm

Åbo Akademi
Institutionen för Informationsteknologi
12.2.2013
Handledare: Mats Aspnäs

Sammandrag

Arbetet behandlar olika teknologier inom programvaruproduktion som stöder horisontell skalning i ett datormoln. Det vill säga distribuerade system som allokerar resurser i molnet enligt användning. Med ett distribuerat system avses ett system som inte begränsar sig till enbart en server, utan kan fördela sig på flera noder. Sådana system är av relevans både för tjänsteleverantörer och programutvecklare som ämnar använda datormolnet som en plattform. Avhandlingen definierar några grundbegrepp väsentliga för området samt presenterar olika teknologiska alternativ inom områdena i fråga. I första hand behandlas olika sorters databassystem och dess egenskaper. Det visar sig att val av databashanterare är högst beroende av tillämpningens karaktär. Vidare beskrivs distribuerade filsystem i allmänhet och Apache HDFS mer ingående. Ett filsystem utgör en grund för andra tillämpningar och det är väsentligt att systemet uppfyller de krav man har på tjänsten. Slutligen diskuteras distribuerade operationer och närmare bestämt Google MapReduce. Denna teknologi möjliggör parallella beräkningar i massiv skala och gör så genom ett enkelt gränssnitt. Detta är en nyckelkomponent i distribuerade tillämpningar där man vill dra nytta av den utökade beräkningskraft som dessa medför. En av avsikterna med arbetet är att lyfta fram de möjligheter en molnmiljö erbjuder, samt att visa att redan existerande teknologier kan utnyttjas för att ta vara på dessa.

Nyckelord: datorbaserade molntjänster, skalbarhet, NoSQL

Innehållsförteckning

1 Inledning

Datorbaserade molntjänster eller ”cloud computing” är en relativt färsk trend inom informationsteknologin. De erbjuder en möjlighet att utan större investering i egen hårdvara kunna erbjuda tjänster på internetskala och betala enligt användning.

Denna avhandling begränsar sig till ett teknologiskt perspektiv utan att ta en konkret ställning till vilken typ av molntjänst det rör sig om. Det vill säga med vilka byggstenar åstadkomma applikationer som dynamiskt kan skalas i en molnmiljö. Det visar sig att många existerande, äldre teknologier kan utnyttjas för att erhålla mera elastiska egenskaper.

Viktiga för att uppnå detta är de underliggande databassystem som driver en applikation. Bland annat presenteras några typiska No-SQL *(”Not only SQL”, dvs. ”Inte enbart SQL”)* databasalternativ . Dessa erbjuder typiskt en större horisontell skalbarhet men har inte kanske alla de gynnsamma egenskaper som en vanlig relationsdatabas har. Det gäller alltså att välja rätt databas för rätt ändamål.

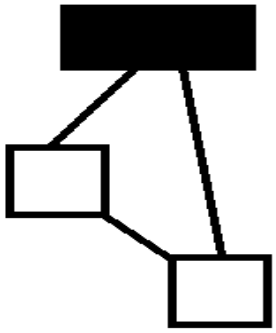
Vidare behandlas kort även distribuerade filsystem, som påminner om om de distribuerade databaserna till en viss grad. Till skillnad från vanliga filsystem stöder de skalning genom replikering, trots att de utåt sett beter sig som ett vanligt filsystem (det vill säga de är *transparenta*).

Till sist behandlas även distribuerade operationer. Detta innebär att data behandlas parallellt på olika servers samtidigt för att sedan pusslas ihop till ett resultat. Detta underlättar tunga operationer som annars skulle ta en lång tid att köra på enbart en maskin.

2 Definitioner

2.1 Skalbarhet

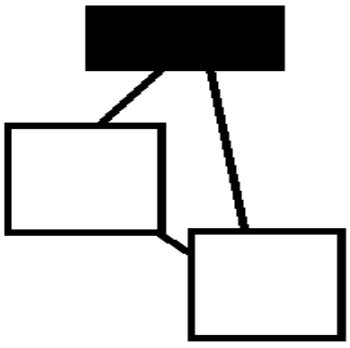
Skalbarhet är ett begrepp som syftar på ett systems förmåga att hantera varierande belastning. I det här sammanhanget avses främst ett växande eller varierande antal av användare. Vidare kan skalningen delas upp i ytterligare två kategorier beroende på vilken strategi som systemet använder sig av för att åstadkomma detta. Figur 1 visar ett generiskt system som används som referenspunkt vid förklaringen av nedanstående begrepp.



Figur 1: Ett generiskt system

2.1.1 Vertikal skalning

Vid vertikal skalning ökas de enskilda nodernas kapacitet för att kunna hantera ett större flöde. Det kan vara frågan om hårdvaruförbättringar så som processor-, primär- eller skivminnesuppgraderingar. Benämningen

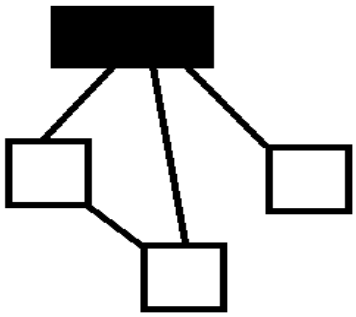


Figur 2: Vertikal skalning

”skala upp” förekommer också. Figur 2 illustrerar vertikal skalning (större noder).

2.1.2 Horisontell skalning

Horisontell skalning innebär att lägga till noder i det befintliga systemet för att fördela bördan, ett så kallat distribuerat system. I detta sammanhang är det närmast frågan om utökat antal virtuella maskiner eller behållare som kör en viss tjänst i molnet. Det vill säga man gör kopior av en applikation eller delar av den för att köra dem parallellt. Denna sorts replikering utnyttjas även på andra nivåer för att förbättra de enskilda applikationernas och själva molnets elasticitet. Alternativt benämningssätt av förfarandet är att ”skala ut”. Figur 3 illustrerar horisontell skalning (större antal noder). [1]



Figur 3: Horisontell skalning

2.2 Molnbaserade datortjänster

Den struktur som ligger bakom molnbaserade datortjänster (*eng. ”cloud computing”*) kommer i fortsättningen att refereras till endast som ett ”moln” (*eng. ”cloud”*). De tjänster som molnleverantören erbjuder är således ”molntjänster”.

Eftersom molntjänster och bakom liggande teknologi för nuvarande är under aktiv utveckling existerar det flera uppfattningar om vad ett moln är. Det som presenteras nedan utgår från en definition av NIST (National Institute of Standards and Technology).

2.2.1 Allmänt

Molntjänster har ett antal typiska egenskaper. Möjlighet att automatiskt kunna skala resurser i molnet enligt klientens krav är bland med de viktigare av dessa. Molnets kapacitet verkar således oändlig från användarens synpunkt. Denna skalning skall kunna ske både uppåt och nedåt. Typiskt mäts och bokförs användningen av resurser enligt förbrukad bandbredd, använt minne eller operativ belastning beroende på vilken typ av service det rör sig om.

Vidare kan molnen klassas i ytterligare kategorier beroende på graden av abstraktion.

2.2.2 Infrastructure as a Service (IaaS)

Användaren har kontroll över allting utom den grundläggande molnstrukturen, som till exempel vilken typ av operativsystem som ska köras. Den underliggande hårdvaran är fortfarande virtualiserad, istället har användaren kontroll över abstrakta servers.

2.2.3 Platform as a Service (PaaS)

Användaren har tillgång till utvecklingsverktyg specifika för molnet i fråga. Applikationerna kan sedan köras på tjänsteleverantörens infrastruktur. Exempel på dylika tjänster är Googles App Engine och Microsofts Azure. En stor fördel med PaaS är att utvecklarna kan fokusera mera på applikationerna och mindre på deras omgivning.

2.2.4 Software as a Service (SaaS)

Klienten använder endast applikationer som körs i molnet utan att ha tillgång till underliggande tekniska detaljer. Typiskt nås dessa applikationer via till exempel en webbläsare. Exempel på tjänster inom denna kategori är bland annat Facebook. [2]

3 Databassystem

Databasen är en central del av webbaserade applikationer och påverkar i stor grad skalbarheten. I en molnmiljö ligger betoningen på horisontell skalning. Detta leder till att distribuerade databassystem är av intresse för att stöda applikationens tillväxtpotential.

Förutom traditionella SQL baserade relationsdatabaser har även andra alternativ dykt upp. Dessa är ofta specialiserade och garanterar inte nödvändigtvis ACID (***A**tomicity, **C**onsistency, **I**ntegrity, **D**urability*) egenskaper. Dock kan det innebära vissa fördelar att ge upp somliga egenskaper för andra. I detta avsnitt beskrivs olika alternativ för att realisera ett distribuerat databassystem, med betoning på olika varianter av *NoSQL.* Även mer konventionella, parallella relationsdatabser presenteras kort som en referenspunkt. [3]

3.1 Allmänt

För förtydligande presenteras kort några grundbegrepp som är av relevans för databashantering.

3.1.1 Rader och kolumner

I en databas lagras information i vad man kan jämföra med en tabell. En rad i en tabell motsvarar en entitet och varje kolumn ett attribut. Attributen är således de samma för alla entiteter i tabellen.

Titel	Artist	Album	År
Breaking the Law	Judas Priest	British Steel	1980
Aces High	Iron Maiden	Powerslave	1984
Kickstart My Heart	Mötley Crue	Dr. Feelgood	1989
Raining Blood	Slayer	Reign in Blood	1986
I Wanna Be Somebody	W.A.S.P.	W.A.S.P.	1984

Figur 4: Exempeltabell [4]

Figur 4 illustrerar en tabell där en låt utgör en entitet och titel, artist, album samt år utgör attribut.

Vanligtvis är systemen radbaserade, det vill säga fokus ligger på entiteterna. Det är lätt att skriva in en ny entitet, den läggs till i slutet av tabellen. Ett kolumnbaserat system fokuserar i sin tur på attributen. I det här fallet är det lättare att handskas med de enskilda kolumnerna, till exempel att summera och räkna medeltalet över ett speciellt attribut.

Slutsatsen är alltså att de olika systemen lämpar sig för olika uppgifter. Ett radbaserat system passar bäst i en miljö med frekventa transaktioner (*OLTP, ”online transaction processing”*), medan kolumnbaserade system utgör en bättre grund för olika analytiska tillämpningar (*OLAP, ”online analytical processing”*).

MySQL är en mycket känd databashanterare som vanligtvis används som ett radbaserat system. Kolumnbaserade system är bland annat Apaches Cassandra och HBase. [4]

3.1.2 Relationsmodellen

Relationsmodellen är en modell för datalagring som ursprungligen definierades av Edgar Codd vid IBM. Enligt modellen organiseras data i tupler som alla har samma attribut. Både tuplerna och attributen är osorterade och samlas vanligtvis i så kallade *relationer* eller tabeller enligt exemplet ovan. Dock måste varje tupel kunna identifieras med hjälp av ett antal attribut, en så kallad *primär nyckel*.

Med hjälp av *SQL (Structured Query Language, ”Strukturerat frågespråk”)* kan sedan olika operationer utföras på tabellerna för att manipulera data.

Andra viktiga begrepp är *främmande nycklar* och *index*. En främmande nyckel är i princip en referens till en annan tupel i en annan tabell. Dessa behöver dock inte vara unika, utan låter ett flertal tupler peka på en och samma tupel. Index är likt nycklar också en samling attribut, men är endast ett sätt för databassystemet att hitta resultat snabbare. [4]

3.2 NoSQL

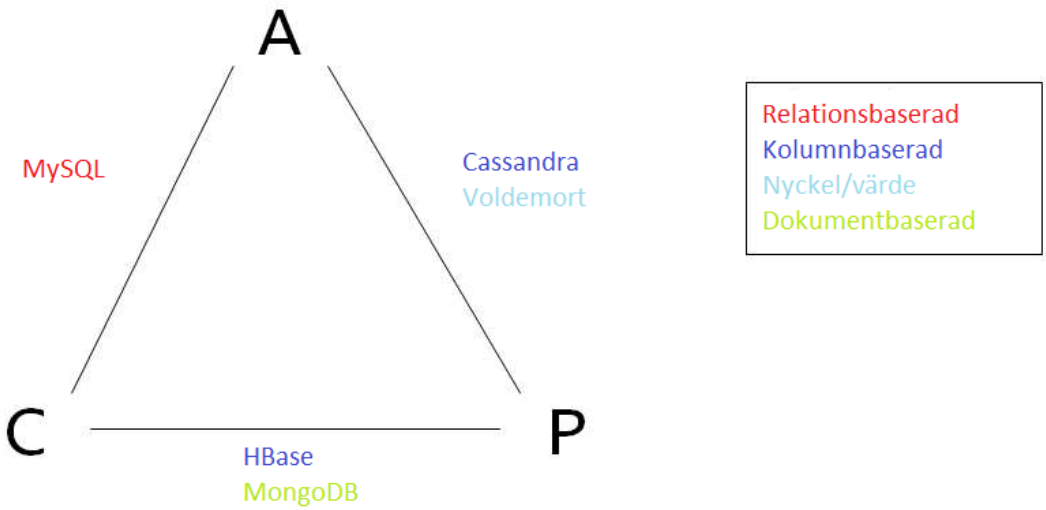
NoSQL eller ”*Not only SQL*” *(”inte enbart SQL”)* är en term som beskriver databaser som inte baserar sig på relationsmodellen. Man lättar i stället på

relationsmodellens krav för att erhålla bättre flexibilitet och prestanda. Trots detta finns det stöd för mer konventionella, parallella relationsbaserade databashanteringssystem även i distribuerade sammanhang. [3]

Typiska egenskaper hos NoSQL-system är att de saknar fasta tabeller och stöd för tabellsammanslagningar. Dessa mekanismer uteslutes för att erbjuda större horisontell skalbarhet. Dessutom är dylika system ofta anpassade för speciella ändamål.

Det visar sig att inget enskilt system kan erbjuda alla ideella egenskaper hos en webbaserad tjänst. Detta kallas för *CAP-teoremet* och presenterades först av Eric Brewer 2000. CAP står för ”***C**onsistency, **A**vailability and **P**artition Tolerance”*, det vill säga konsistens, tillgänglighet och partitionstålighet. Teoremet säger att alla tre omöjligen kan garanteras samtidigt.

Konsistens innebär att operationer utförs i sin helhet eller inte över huvudtaget. Detta erbjuds ofta i formen av transaktioner. Med tillgänglighet menas att systemet svarar på varje mottagen förfrågan. I ett distribuerat system kan det uppstå partitioner. Detta innebär att det existerar grupper av noder i systemet vars tillstånd är i konflikt med varandra.



Figur 5: Olika databashanterares relation till CAP-teoremet [4]

Att undvika partitioner innebär i praktiken att utelämna den distribuerade aspekten av systemet och endast använda sig av en enda server. Förstås

leder detta till dålig skalbarhet överlag. Att ge upp tillgänglighet betyder i verkligheten att tjänsten väntar då det uppstår partitioner. Om däremot konsistens ignoreras fungerar allt det ovanstående förutom att data inte nödvändigtvis är helt korrekt. Till exempel behöver innehållet i inköpskorgen för en webbtjänst inte motsvara det egentliga. Däremot behöver kunder nödvändigtvis inte utsättas för dylika incidenter om viktiga delar av systemet garanterar *ACID* egenskaper, medan andra kanske endast är *BASE (**Ba**sically available, **S**oft state, **E**ventually consistent).* Figur 5 illustrerar olika databashanterares relationer till CAP-teoremet. [4]

3.2.1 Nyckel/Värde-databaser

Nyckel/värde-databaser baserar sig på en enkel design där data lagras i par om en nyckel och ett värde i en räcka. Nyckel/Värde-system stöder endast enkla operationer som att hämta, uppdatera eller lägga till en entitet. I grund och botten beter sig systemet som en stor hashtabell. Exempel på Nyckel/Värde-databaser är Project Voldemort och Amazons SimpleDB. [4]

3.2.2 Dokumentdatabaser

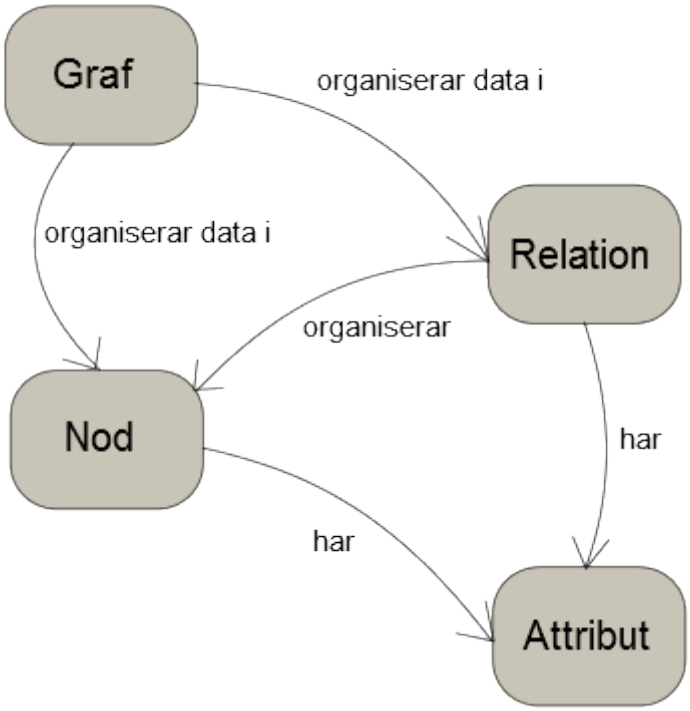
En dokumentbaserad databas lagrar till skillnad från traditionella relationsdatabaser inte data i tabeller utan kopplar ihop ett godtyckligt antal attribut av valfri storlek i ett så kallat dokument. Fördelen med detta att användaren kan modifiera attributen i en tupel utan behöva lägga till tomma attribut till alla sammanlänkade tuplar. Utöver detta existerar vanligtvis koncepten nyckel och index på samma sätt som i en relationsdatabas. Dokumentbaserade databaser kan implementeras ovanpå en relationsbaserad miljö eller direkt i till exempel XML. Exempel på dokumentbaserade databaser är MongoDB och CouchDB. [4]

3.2.3 Grafdatabaser

Grafdatabaser baserar sig som namnet antyder på en graf. Grafer i sig är mycket generiska strukturer och kan representera data på ett intuitivt sätt. Nyckel/värde-, tabell- och dokumentdatabasers innehåll kan således alltid

tolkas som en graf.

Strukturen i sig består av två typer av objekt: noder och relationer. Noderna representerar typiskt entiteterna och relationerna kopplar ihop dem i ett nätverk av information. Utöver dessa kopplingar har typiskt både noderna och relationerna olika attribut som beskriver dem.



Figur 6: Exempel på en grafs struktur [5]

Figur 6 illustrerar en grafs grundläggande uppbyggnad. Grafen har fyra olika noder med endast ett attribut var (nodens namn i det här fallet). Relationerna är fem till antalet och har likväl endast ett attribut som beskriver relationens namn. En graf kan med hjälp av sina relationer motsvara en rad olika andra datastrukturer, bland annat listor, tabeller och träd.

Förfrågningar görs genom att traversera grafen på olika sätt. På så sätt undviks beräkningsmässigt intensiva tabellsammanslagningar. Dessutom indexeras både noder och relationer för snabbare tillgång. Exempelvis skulle förfrågningar som hämtar noder som noden Graf organiserar data i effektivt kunna utföras genom att hämta de noder som relationer med

namnet ”organiserar data i” pekar på.

Grafdatabaser har flera fördelar gentemot traditionella relationsdatabaser. De är mycket flexibla; nya relationer kan lätt läggas till utan större modifikationer. Dessutom är komplexa förfrågningar som endast berör enstaka entiteter effektivare i en grafmiljö. Således lämpar sig grafbaserade databaser bra för till exempel sociala nätverk och andra liknande strukturer. [6]

Neo4j är ett exempel på en grafbaserad databashanterare. [5]

3.3 Parallella relationsdatabaser

Vanliga relationsbaser kan spjälkas upp i delar, så kallad partitionering eller *sharding (”fragmentering”).* Detta görs genom att del upp databasen per tabell eller genom att spjälka tabellerna horisontellt. Fragmenten kan sedan köras på olika databasinstanser. Detta medför vissa komplikationer, till exempel när transaktioner omfattar flera fragment. Hur man bäst delar upp tabellerna kan också bli ett problem. För MySQL existerar det tredje partens programvara som löser dessa problem för användaren, till exempel HarvestDB. [7] Programvaran fungerar utåt sett som en vanlig relationsdatabas men sköter internt om fragmenteringen. Googles databas F1 däremot, klarar av fragmentering utan extra programvara och kan klassas som en vanlig relationsdatabas med stöd för konventionella SQL-förfrågningar. [8]

4 Distribuerade filsystem

Distribuerade filsystem (eller nätverksfilsystem) är filsystem designade att fungera över ett nätverk av filservers och ge gemensam tillgång till resurserna i filsystemet. Utåt sätt fungerar de vanligtvis som ett vanligt filsystem, det vill säga den underliggande funktionaliteten göms undan. Filsystemet är *transparent*. I en molnmiljö förefaller ett distribuerat filsystem vara ett naturligt val. Dessutom kan filsystemet sköta om replikering av data för att garantera tillgänglighet. Eftersom det är frågan om ett gammalt koncept finns det många olika implementationer med olika

egenskaper. Apaches Hadoop Distributed File System och Amazons S3 är exempel på distribuerade filsystem.

4.1 Hadoop Distributed File System (HDFS)

HDFS är en del av Apaches Hadoop projekt och är designat för att köras på billig hårdvara, där risken för fel är signifikant. Detta förutsätter en mycket feltolerant struktur. Fokus ligger på dataflöde snarare än reaktionstider. Således lämpar sig HDFS för applikationer som jobbar med stora datamängder.

4.1.1 Målsättningar

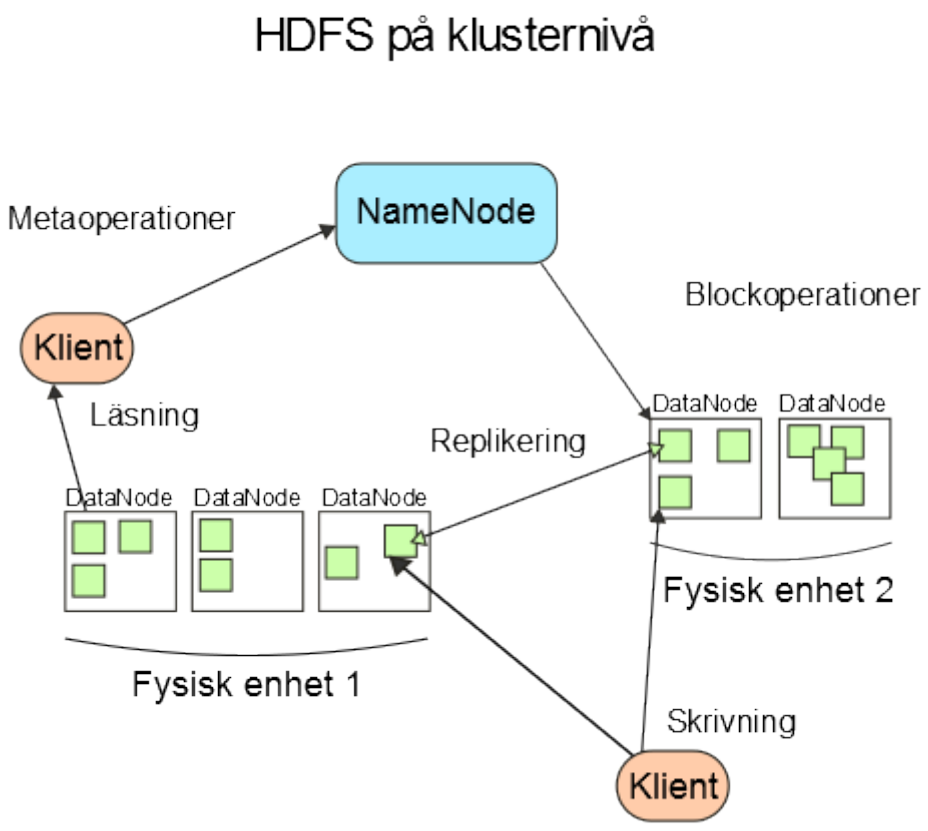
Fel i distribuerade filsystem är regel snarare än undantag. Därför satsar man i HDFS på att gardera sig mot dem snarare än att undvika dem helt och hållet. HDFS är inte designat för att användas med typiska applikationer, utan med tillämpningar som hanterar stora strömmar av data. Vidare baserar sig systemet på en modell som går ut på sporadisk skrivning men riklig läsning av data. En annan intressant aspekt är att flyttande av beräkningar prioriteras över flyttande av data. Detta för att minimera informationsflödet. Med hjälp av de mekanismer som HDFS erbjuder kan en applikation således flytta sig själv närmare den information som skall hanteras. Slutligen poängteras portabilitet; HDFS är skrivet i Java och kan köras på en mängd olika maskiner. [9]

4.1.2 Implementation

HDFS bygger på en master/slave-arkitektur där noderna delas in i kluster. I ett kluster finns en så kallad NameNode *(i forts. ”namnnod”)* som styr de övriga noderna i klustret. En DataNode *(i forts. ”datanod”)* är en nod som enbart hanterar enskilda block av data.

HDFS använder sig av en konventionell hierarkisk namnrymd, vilken upprätthålls av namnnoden. Systemet stöder de vanligaste filoperationerna, men dock ej användarkvoter eller tillgänglighetsrestriktioner.

Replikering är en väsentlig del av systemets tillgänglighets- och säkerhetspolicy. Antalet kopior en fil bör sparas i kallas filens replikeringsfaktor och bokförs per fil i namnnoden. Alla filer delas upp i lika stora block (förutom det sista blocket). Dessa kan enskilt replikeras tills de uppnår den önskade replikeringsfaktorn. Alla operationer på blocken kontrolleras av namnnoden, som också sparar information om blockstorlek och antalet kopior per block. Datanoderna är skyldiga att underrätta namnnoden om deras status samt att skicka blockrapporter med jämna mellanrum. Detta sker genom att noderna periodvis skickar ett så kallat *heartbeat ("hjärtslag")* till namnnoden. Utöver detta skickas även en lista över alla block som datanoden för tillfället lagrar.



Figur 7: Kluster i HDFS [7]

Placering av de olika replikeringarna är viktigt för systemets funktionalitet och är en icke-trivial uppgift. Det mest intuitiva skulle vara att placera alla kopior i skilda rackar eller fysiska enheter. Detta ifall att en rack helt och hållet sätts ur funktion. Däremot är bandvidden inom en rack vanligtvis

större än den mellan olika rackar. Dessutom är chansen att en enskild nod slås ut betydligt större än en hel rack. Dessa två faktum bidrar till att den nuvarande replikeringspolicyn med en faktor om tre skulle placera två enheter på olika noder inom samma rack, samt en enhet på en extern rack. Detta visar sig förbättra hastigheten på skrivoperationer utan att innebära kompromisser i form av nedsatt läshastighet eller tillgänglighet. Det är alltså frågan om ett optimeringsproblem med säkerhet och prestanda som faktorer.

Vidare försöker systemet optimera prestandan genom att vid läsning och skrivning välja den kopia som ger bäst resultat. Detta beteende följer ett slags närhetsprincip och prioriterar replikeringar som existerar i samma rack före kopior i samma datacenter. Sist prioriteras kopior på en global nivå.

Viktigt att notera är att namnnoden endast sköter administrativa uppgifter, själva dataströmmarna vid läsning och skrivning öppnas direkt mellan klienten och datanoden i fråga. Ett kluster kan omfatta flera rackar eller till och med flera datacenter. Detta illustreras i figur 7, som avbildar ett kluster och dess inbördes relationer.

Namnnoden kan befinna sig i ett speciellt säkerhetstillstånd, ”s*afemode”,* till exempel vid systemuppstart. I detta läge är all replikering av block mellan olika noder förbjuden. Istället väntar namnnoden på att datanoderna internt uppnår en viss replikeringsgrad. Efter detta återgår namnnoden till sitt vanliga tillstånd. I detta skede kopieras de filer som ej ännu uppnått önskad replikeringsgrad till externa noder.

Utöver tidigare nämnda standardreplikering kan klustret dynamiskt generera extra kopior av block eller hela filer som har en hög efterfrågan. Dessutom kan klustret internt flytta på replikeringar för att balansera eventuella belastningsskillnader. [9]

4.1.3 Kommentarer

Med bland annat ovannämnda replikeringsmekanismer försöker HDFS garantera en kombination av prestanda och säkerhet. Systemet i sig täcker inte alla säkerhetshål till fullo. Ett fel i namnnoden kan till exempel

orsaka stora problem eller till och med förlust av data. HDFS erbjuder dock mekanismer som stöder horisontell skalbarhet och fungerar som en grund för tillämpningar som hanterar stora datamängder samt kräver hög skriv- och läsprestanda. Konceptet med nätverksfilsystem är i sig gammalt och andra alternativ med varierande egenskaper existerar.

5 Distribuerade operationer

Med distribuerade operationer avses beräkningar som kan delas upp och köras parallellt. Detta är synnerligen viktigt då stora mängder data måste behandlas och man vill utnyttja de resurser ett datacenter erbjuder. Detta är i sig ingenting nytt, utan det existerar flera alternativa implementationer. Bland annat Apaches Hadoop MapReduce och Googles MapReduce, som båda implementerar MapReduce-programmeringsmodellen.

5.1 MapReduce

MapReduce är en abstrakt programmeringsmodell som gömmer undan de problem och aspekter som distribuering medför. Istället definierar programmeraren två funktioner för ett givet problem: *map* (dela upp) och *reduce* (reducera). Namnen är inspirerade av map- och reducefunktionerna i funktionella språk, dock med annan funktionalitet. Som namnet säger arbetar metoden genom att dela upp problemet i mindre bitar för att sedan slå ihop resultaten. Det här arbetet refererar till Googles implementation av MapReduce.

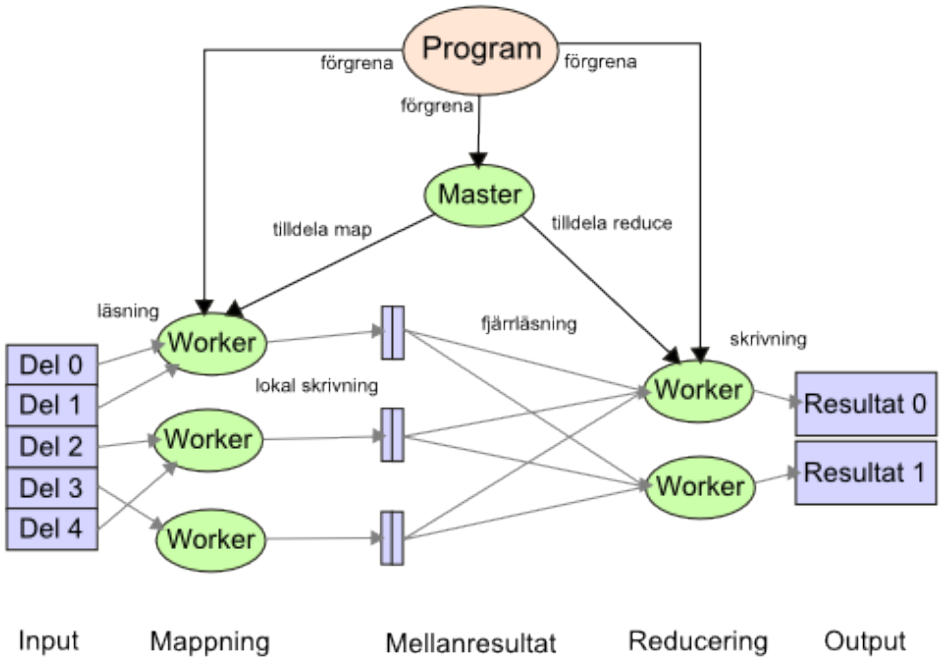
5.1.1 Funktionsprincip

En MapReduce beräkning kan delas in i ett antal steg. Hela processen illustreras i figur 8. I det första steget distribueras applikationen till ett antal noder. Utav dessa utses endast en till *master (”mästare”).* Mästaren fungerar som en metanod som styr hela processen. Resten av noderna fungerar som *workers (”arbetare”).*

Den data som skall behandlas delas i ett antal delar, typiskt mellan 16 och 64 MB stora. Därefter delar mästaren ut uppdrag åt arbetarna. Dessa kan

vara av två typer: map eller reduce. Totalt delas M respektive R stycken map- och reduceuppdrag ut.

En arbetare som tilldelats ett mapuppdrag läser information från de delar av källan som associerats med ifrågavarande arbetare. Informationen transformeras till par bestående av en nyckel och ett värde. Dessa par körs sedan igenom användarens mapfunktion Resultatet av mappningen skrivs till den lokala hårdskivan med jämna mellanrum. Denna är uppdelad i R stycken partitioner, en för varje reduceuppdrag. Adresserna för dessa nyckel-värdekombinationer skickas vidare till mästaren som i sin tur vidarebefordrar dem till motsvarande reducearbetare.



Figur 8: Funktionsprincip hos MapReduce [10]

Reducearbetaren hämtar mellanresultatet från maparbetarens lokala hårdskiva via fjärrproceduranrop. Informationen lagras och sorteras enligt nyckel. Programmet kör användarens reducefunktion för varje unik nyckel och itererar i denna igenom sammanhängande värden. Resultatet av funktionen sparas i en för arbetaren specifik partition.

Mästaren har under körningen som uppgift att hålla reda på varje uppdrag och dess status. Utöver detta förmedlar mästaren vidare mellanresultatets olika platser och storlekar till reducearbetarna. Denna information uppdateras givetvis an efter att mappningsuppdragen slutförs.

Till sist när alla uppdrag slutförts väcker mästaren upp användarens program, som fortsätter körningen som vanligt efter MapReduce-anropet. Resultatet finns lagrat i R stycken olika filer, som kan utnyttjas som sådana eller skickas vidare till ett nytt MapReduce-uppdrag. [10]

5.1.2 Feltolerans

Precis som i fallet HDFS är MapReduce designat att köras på helt generisk hårdvara där fel och andra störningar är ett faktum. Detta speciellt i stora kluster som innehåller tusentals datorer.

Det mest sannolika scenariot är att någon av arbetarna misslyckas i sin exekvering av programmet. Mästaren detekterar dylika händelser genom att med specifika intervaller testa kontakten till varje arbetare. Om ingen respons erhålles, markerar mästaren arbetaren som misslyckad. Varje uppdrag som slutförts av en maparbetare måste markeras som ogjorda och köras på nytt. Detta för att resultatet av dessa uppdrag lagras på arbetarens egen lokala hårdskiva. Alla reducearbetare måste notifieras om händelsen för att kunna läsa data från rätt nod. Reduceuppdragens resultat lagras däremot i ett globalt filsystem och behöver inte genereras på nytt.

Mer sällsynt är att mästaren skulle dö. I det fallet att mästaren regelbundet gör kopior av sina datastrukturer i det globala filsystemet kan en ny kopia lätt startas. Dock är detta så osannolikt att Googles implementation helt enkelt avbryter hela MapReduce exekveringen om mästaren förloras. På så vis måste istället hela processen startas om.

Om användarens map- och reducefunktioner är deterministiska, det vill säga alltid producerar samma utvärde för ett givet invärde, garanterar systemet att resultatet av hela den distribuerade exekveringen är ekvivalent med en sekventiell motsvarighet. [10]

5.1.3 Kommentarer

MapReduce gör det möjligt att inom en rimlig tidsrymd utföra beräkningar på annars överväldigande mängder data. Typiskt för de olika

implementationerna är ett enkelt användargränssnitt kombinerat med en kraftfull distribuerad algoritm. I materialet som använts i beskrivningen ovan meddelas att Google typiskt kör MapReduce-beräkningar med 200000 map- respektive 5000 reduceuppdrag. Detta uppdelat på 2000 arbetare. Summa summarum erbjuder ramverket ett enkelt sätt att dra nytta av stora kluster av datorer vid beräkningsmässigt intensiva operationer.

6 Slutsatser

Vad arbetet har visat är att det existerar många teknologiska alternativ vid implementation av tjänster som kräver skalning på ett horisontellt plan. Vid val av dessa underliggande funktioner bör tjänstens natur noga begrundas, då teknologierna trots sina gemensamma skalningsmöjligheter även har väldigt individuella egenskaper. Detta gäller speciellt för databaser, där valet av teknologi direkt har en betydelse för tjänstens funktionalitet. Databasens förhållande till CAP-teoremet eller dess datamodell berättar i sig mycket om vilka användningsområden som lämpar sig.

En webbtjänst som hanterar stora mängder statistiska data samtidigt som portioner av sensitiva användardata kan till exempel använda sig av två helt skilda databassystem för vardera ändamålet. I det första fallet kanske inte konsistens är viktigt och man kan gott utelämna denna egenskap för att erhålla mera prestanda. Det andra fallet kanske kräver ACID-transaktioner men kan slopa tillgänglighetskravet. Detta om korrektheten är viktigare än omedelbar tillgänglighet.

Vidare är sättet på vilka data skall behandlas avgörande för vilken typ av databas som är att föredra. System som använder sig av komplexa förfrågningar som endast berör en liten andel av innehållet kan exempelvis dra nytta av en grafdatabas och dess egenskaper. Den andra ytterligheten är simpla operationer på stora mängder data, där en databas med striktare struktur klarar sig bättre. En kolumnbaserad databas lämpar sig väl i statistiska och analytiska tillämpningar.

Filsystemet är i sig en viktig komponent i ett distribuerat system eftersom det utgör den grund som många andra tillämpningar förlitar sig på. Replikering inom klustret eller också mellan olika datacenter är egenskaper man kan förvänta sig av en molntjänst. Detta fungerar samtidigt som säkerhetskopiering också som ett sätt för systemet att dela upp belastningen mellan olika helheter. I vissa tillämpningar kan det också vara viktigt att data finns tillgängligt geografiskt nära användaren, något som också kan lösas genom replikeringsmekanismen.

Tjänster som hanterar stora datamängder åt gången och som beror av sin distribuerade omgivning för att vara effektiv, kan dra nytta av ramverk så som MapReduce. Detta för att lätt kunna behandla stora mängder data till exempel vid omfattande sökningar eller andra tunga beräkningar.

Vad allt detta visar är att trots att molnbaserade datortjänster i sig är ett relativt nytt område, kan dessa tillämpningar dra nytta av redan existerande teknologier, som i sig inte nödvändigtvis är direkt utvecklade för molnet.

Källförteckning

[1]: R. Anandhi - K Chitra, A Challenge in Improving the Consistency of Transactions in Cloud Databases - Scalability, International Journal of Computer Applications, No. 2, Vol. 52, 2012

[2]: Mell, Peter - Grance, Timothy, The NIST Definition of Cloud Computing, http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, hämtad: 19.02.2013

[3]: Stonebraker, Michael, SQL Databases v. NoSQL Databases, Communications of the ACM, No. 4, Vol. 53, 2010

[4]: Lith, Adam - Mattson Jakob, Investigating storage solutions for large data, Chalmers University of Technology, 2010

[5]: The Neo4j Team, The Neo4j Manual v1.9.M05, http://docs.neo4j.org/, hämtad: 18.03.2013

[6]: Batra, Shalini - Tyagi, Charu, Comparative Analysis of Relational and Graph Databases, International Journal of Soft Computing & Engineering, No. 2, Vol. 2, 2012

[7]: Humbug Telecom Labs Ltd, HarvestDB, http://www.harvestdb.com, hämtad: 29.03.2013

[8]: Shute, Jeff et Al., F1 - The Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business, http://www.stanford.edu/class/cs347/slides/f1.pdf, hämtad: 29.03.2013

[9]: Borthakur, Dhruba, The Hadoop Distributed File System: Architecture and Design, http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf, hämtad: 20.03.2013

[10]: Dean, Jeffrey - Ghemawat, Sanjay, MapReduce: Simplified Data Processing on Large Clusters, Communications of the ACM, No. 1, Vol. 51, 2008

Figurförteckning