Zodra applicaties zijn gedeployd starten de problemen: Meerdere gebruikers maken verbinding met de applicatie en klagen over de snelheid. Waar begin je om het probleem te lokaliseren? Is het het besturingssysteem? Hoe zit het met de garbage collection? Is de architectuur van de applicatie een probleem? We geven een startpunt hoe een probleem te analyseren en daarnaar te handelen. Daarbij kunnen we niet alle aspecten van Java EE performance behandelen. We presenteren een aantal best practices en tips.

Tune de performance

Doe er ervaring mee op en heb er voordeel van

Voordat we ons gaan verdiepen in performance tuning, is het belangrijk een overzicht te hebben van de interne architectuur van de WebLogic Server. Het verwerken van requests gaat aan de hand van de volgende componenten: listen threads, de socket muxer en de execute queue. Wanneer de server opstart, koppelt de server listen threads aan elke geconfigureerde poort. De socket muxer detecteert een binnenkomend request en plaatst deze in de execute queue. Een vrije execute thread pikt het request uit de queue op en voert deze uit. De execute thread voert het hele request uit, met andere woorden de aanroep naar een servlet, de servletaanroep naar een EJB en de EJB aanroep naar JDBC om een databasequery uit te voeren, worden afgehandeld door één execute thread.

Een belangrijke constatering is dat als applicatiecode een execute thread voor een lange tijd blokkeert, de server deze execute thread niet kan gebruiken om andere requests te verwerken. Als de applicatie in een toestand komt waarin elke thread voor een onbepaalde tijd is geblokkeerd, reageert de server niet meer of er worden additionele threads aangemaakt. Het aanmaken van nieuwe threads die uiteindelijk ook worden geblokkeerd helpt natuurlijk de gehele situatie ook niet. Dat threads niet in afzienbare tijd worden verwerkt, komt de performance niet ten goede.

Nu we een klein beetje een idee hebben wat er intern gebeurt gaan we eens kijken naar systeem performance.

Systeem Performance

Een goed begrip van het operating systeem, netwerk, JVM en de gebruikte server resources met bijbehorende tuning opties helpen enorm bij het toepassen van best practices. In het algemeen is het niet voldoende wat je moet doen, je moet ook weten waarom het werkt.

Als we applicaties ontwerpen, moeten we eerst de applicatie zelf begrijpen en hoe de gebruikers ermee omgaan. We moeten alle systeemcomponenten in kaart brengen en de interactie tussen deze componenten begrijpen. Met het in kaart brengen van de workload over alle lagen begrijpen we welke componenten er het meest worden beïnvloed door de activiteit van gebruikers. Het goed begrijpen van het systeem stelt ons in staat de juiste systeemarchitectuur te kiezen. Als eenmaal de systeemarchitectuur vast ligt, kunnen we ons gaan focussen op de architectuur van de applicatie.

Als er na wel overwogen beslissingen in productie toch performanceproblemen optreden of we willen het maximale uit de server halen, moeten we gaan tunen. Het tunen gebeurt op verschillende lagen binnen de omgeving. De volgende discussie begint op de bodem en vervolgens werken we ons gestaag naar boven.

Tunen van het operating systeem

Java EE applicaties hebben in het algemeen één of andere webinterface. In het algemeen hebben applicaties een paar duizend gelijktijdige gebruikers. Met als gevolg dat er een hoog aantal connecties tussen de browser en de server worden geopend en gesloten. Deze connecties zijn niets anders dan TCP sockets op operating systeemniveau. De meeste operating systemen behandelen sockets als een vorm van file access en maken gebruik van file descriptors om bij te houden welke sockets open staan.

René van Wijk
is consultant, trainer bij Transfer Solutions.

Het is erg belangrijk te weten hoe de JVM de garbage collecion uitvoert.

Om resources in de perken te houden voor andere processen op een machine beperkt het operating systeem het aantal file descriptors per proces.

Alle TCP connecties die netjes zijn gesloten door een applicatie bevinden zich in de toestand TIME_WAIT voordat ze worden teruggegeven aan het operating systeem. Zolang de connectie zich in de toestand TIME_WAIT bevindt, blijven alle resources gealloceert (inclusief de file descriptor). Met als gevolg dat de file decriptor tabel vol kan raken. In het algemeen betekent dit dat we het operating systeem moeten tunen, zodat een schaalbare applicatie niet tegen een restrictie van het operating systeem aanloopt. Het tunen omvat meestal het volgen van aanbevelingen van de hardware vendor.

Tunen Java Virual Machine (JVM)

De JVM die wordt gebruikt om de server met bijbehorende applicaties te draaien is van enorm belang in de uiteindelijke performance van de server. Wat we in het achterhoofd moeten houden, is dat performance niets is als we geen stabiliteit hebben - gebruikers hebben niets aan snelle applicaties als ze niet draaien. Dus bij de keuze van een bepaalde JVM is stabiliteit een pré, daarna de performance.

De garbage collection is de belangrijkste factor bij het tunen van een JVM. Slecht getunede garbage collectors of applicaties die onnodige hoeveelheden objecten aanmaken kunnen dramatische gevolgen hebben voor de performance van de applicatie. Het juist tunen van de garbage collector reduceert de processing tijd enorm met als gevolg een flinke verbetering van de applicatieperformance.

Net zoals het belangrijk is om de workload van de applicatie te weten, is het eveneens belangrijk te weten hoe de gebruikte JVM de garbage collection uitvoert. Als we eenmaal een goed begrip hebben van de garbage collection algoritmes, is het mogelijk de applicatie en garbage collection te tunen, zodat de performance maximaliseert. Meer informatie hoe garbage collection algoritmes van invloed zijn op de performance is te vinden in het artikel 'Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory' van Nagendra Nagarajayya en Steve Mayer.

Tunen van de server

Tijdens het testen van de performance is het belangrijk de listen queue te vergroten. WebLogic Server gebruikt de Accept Backlog parameter om de grootte van de queue te specificeren. Als client requests worden afgewezen kan het zijn dat de grootte van de queue te klein is. De parameter Login Timeout wordt gebruikt voor het specificeren van de maximale tijd om een connectie op te zetten. Standaard is deze waarde 5 seconden (voor SSL is deze 25 seconden), wat te klein kan zijn voor systemen met een zware load.

Het optimaliseren van het thread management wordt door de WebLogic Server zelf afgehandeld. Door het verzamelen van performance-informatie wordt het aantal execute threads aangepast aan de workload van de applicatie. Standaard maakt elke applicatie gebruik van een default work manager. Hiermee krijgt elke applicatie dezelfde prioriteit en voorkomt dat applicaties meer dan hun deel van de server resources kunnen claimen. In het algemeen is de default work manager voldoende.

Er zijn echter een aantal punten die van belang zijn:

- Database connectiepool - Als de applicatie afhangt van database connecties, is het van belang in acht te nemen dat er niet meer gelijktijdige requests verwerkt worden dan dat er connecties in de pool zijn. In dit geval kunnen we de default work manager overriden door een maximum thread constraint op te leggen. Deze constraint zetten we gelijk aan het aantal connecties in de pool.
- Server deadlock - Sommige applicaties roepen resources aan in een andere server instantie, die op zijn beurt resources aanroept in de aanroepende server. In dit geval moeten moeten we extra zorg nemen zodat een server deadlock niet voorkomt. De deadlock treedt op als alle threads staan te wachten op requests van de remote applicatie en er geen thread beschikbaar is om de callback te verwerken. In dit geval moeten we een minimum thread constraint configureren, zodat er ten alle tijden threads over zijn om de callback te verwerken.
- Server overload - Als we de server tunen om een bepaalde response tijd te halen, is het belangrijk in te zien dat het op een bepaald punt geen zin meer heeft nog nieuwe requests te accepteren. Als een request zich in de queue bevindt, heeft deze een bepaalde tijd nodig voordat deze verwerkt kan worden (het request staat in de wachtrij te wachten op een execute thread). Om er voor te zorgen dat de queue niet overvol raakt, configureren we een capacity constraint. Als de capaciteit wordt overschreden wordt er een 503 response verstuurt.

De WebLogic Server maakt gebruik van resource pooling om de performance te optimaliseren. Zo worden onder andere stateless session beans en message-driven beans gepoold. Als alle resources worden gebruikt, zal de server de pool vergroten om zo aan de requests van de applicaties te voldoen. Het verhogen van de pool voegt overhead toe aan het verwerken van een request. De over-

Er kunnen meestal verbeteringen gehaald worden bij de garbage collection….

head hangt sterk af van type resource. Als de pool zijn maximum bereikt, kan de server de pool niet meer vergroten. Als deze situatie zich voordoet, moeten requests wachten totdat er een resource vrijkomt voordat het request kan worden verwerkt wat de performance de nek om draait. Een voor de hand liggende vuistregel die gehanteerd moet worden is: ervoor zorgen dat de pool groot genoeg is voor het te verwerken van het aantal gelijktijdige requests.

Als eenmaal de optimale instellingen gevonden zijn, kunnen er meestal nog wat verbeteringen worden gehaald bij de garbage collection.

Voordat een applicatie in productie wordt gezet, is het van belang langdurige load tests uit te voeren. Deze tests kunnen in het algemeen problemen aan het licht brengen, die in productie pas na een lange tijd boven komen drijven, zoals bijvoorbeeld geheugenlekken.

Performance in het algemeen

Een juiste architectuurkeuze (de juist patronen) kan de performance verbeteren. Het begrijpen waarin de applicatie draait, zoals de web en de EJB container en wat deze zoal te bieden hebben is voor een juiste opzet van de architectuur van enorm belang.

Ontwerp

Goede performance van een applicatie begint bij een goed ontwerp. Te complexe applicaties zullen altijd een slechte performance hebben, wat je ook probeert aan tuning. Een schitterend boek over ontwerp patronen of beter gezegd anti-patronen is het boek ‘Bitter Java’ van Bruce Tate.

Het juist gebruik van ontwerppatronen kan significante voordelen opleveren, zoals het standaardiseren van algemene ontwerpproblemen. Bepaalde ontwerppatronen leveren bovendien performanceverbeteringen op, bijvoorbeeld het session façade patroon en het command patroon.

Session façades verbeteren de performance van een applicatie door het aanbieden van alleen high-level business operaties. Dit is zeker van belang bij het aanroepen van EJB’s door middel van remote interfaces (netwerk overhead). Zelfs bij het aanroepen van lokale interfaces is er een overhead, doordat de EJB container, security, lifecycle management en transactie management services verzorgt.

Het command patroon maakt gebruik van één command object om requests als één aanroep te verwerken. Door dit patroon samen te gebruiken met het session façade patroon worden remote aanroepen

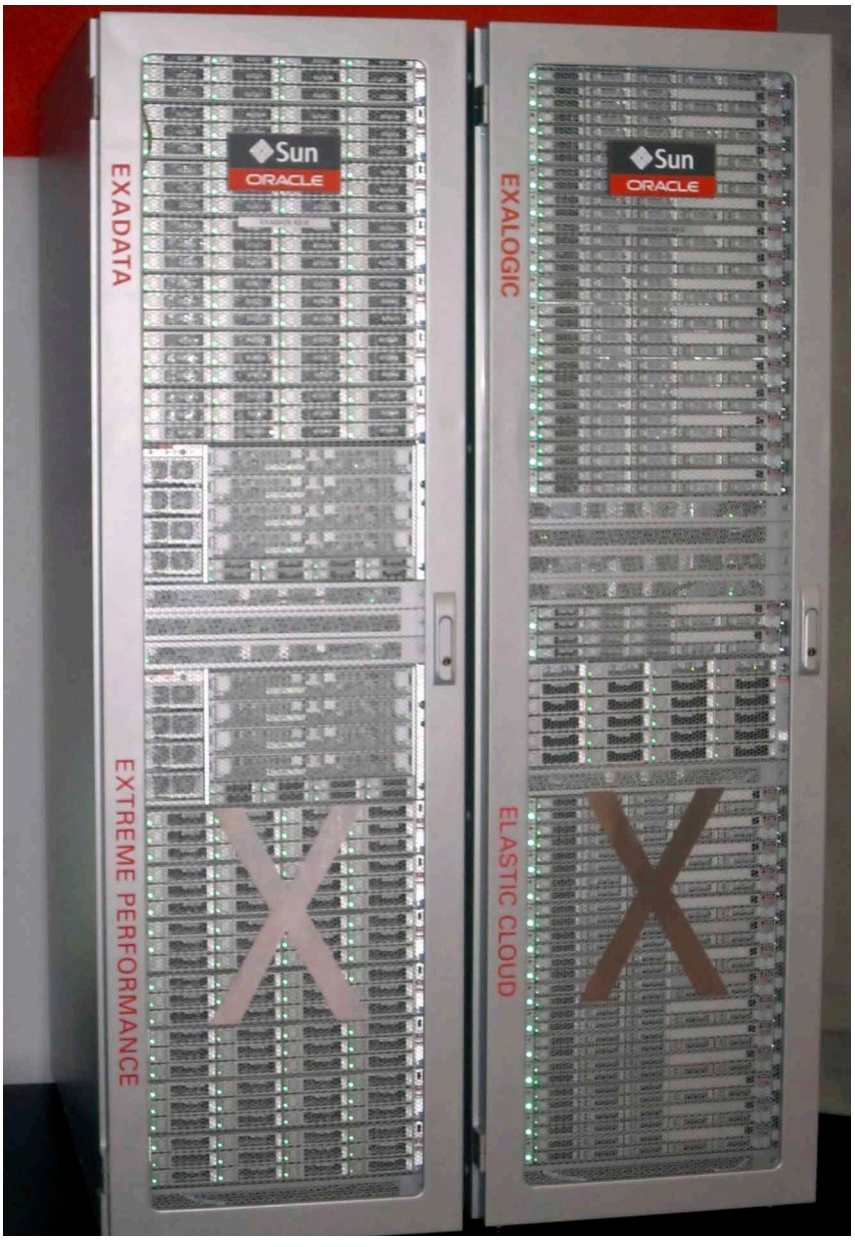
Voordat een applicatie in productie gaat, moeten langdurige load tests worden uitgevoerd.

Een juiste keuze van de architectuur kan tunen van de performance makkelijker maken.

nog verder verminderd. In het boek 'Java Persistence with Hibernate' van Christian Bauer en Gavin King wordt in het hoofdstuk 'Creating and testing layered applications' een goed voorbeeld gegeven.

Web container

Veel applicaties vullen het HTTP protocol aan door objecten op te slaan in het HttpSession object. Deze sessiedata is voor dezelfde browsersessie beschikbaar in vervolgrequests. In een betrouwbare applicatie moet de sessiedata beschikbaar zijn zelfs als de originele server faalt. Dit houdt in dat de sessiedata gerepliceerd moet worden naar andere servers binnen het cluster. Hou er echter rekening mee dat er een kostenplaatje aan hangt om sessiedata te repliceren. Als een webapplicatie het HttpSession object aanpast, moet de server deze veranderingen aan het einde van elk request opslaan. De hoeveelheid data die de server moet opslaan ligt aan de datastructuur in het HttpSession object. Als de webapplicatie een kleine hoeveelheid data verandert in een grote objectstructuur, is het van belang de objectenstructuur op te delen in kleine stukjes. Een HttpSession object gebruikt ook resources dus is het belangrijk deze op te ruimen als je ermee klaar bent.



Als deze hardware performanceproblemen geeft, is de kans groot de code er als spaghetti uitziet.

EJB container

EJB componenten kunnen een dramatisch effect hebben op de performance, omdat deze in het algemeen meer aan het werk zijn dan een webcomponent. We kunnen de aanroep naar EJB's optimaliseren. Als de aanroeper en de EJB zich in dezelfde JVM bevinden kunnen we met de WebLogic Server ervoor zorgen dat er een pass-by-reference optimalisatie plaatsvindt. Deze optimalisatie moeten we voor elke EJB expliciet aanzetten in de deployment override weblogic-ejb-jar.xml (zet het enable-call-by-reference element van een bepaalde EJB op True).

Databasetoegang

Een efficiënte databasetoegang is van levensbelang om een hoge throughput en goede schaalbaarheid te halen. Al het andere tunen is eigenlijk voor nop als de database toegangtraag is.

Ten eerste moeten we een efficiënt logisch databaseontwerp hebben. Een klassieker op dit gebied is 'An Introduction to Database Systems' geschreven door C.J. Tate. Veel database-ontwerpers hebben een voorkeur voor veel normalisatie in hun ontwerp. Dit heeft als gevolg dat er multi-way joins nodig zijn om data op te halen voor een business object. Een ontwerp dat er leuk uitziet op papier, wordt in het algemeen een ramp in productie. Als performance een hoofdcriterium is, wordt het eens tijd om met de database-ontwerper om tafel te gaan zitten en het ontwerp voor kritische tabellen te denormaliseren.

De volgende stap is dat de fysieke database de performance-eisen moet halen. Een DBA moet alle beschikbare optimalisatie in acht nemen om de best mogelijk performance te bereiken. Hier is het van belang dat de DBA een complete lijst heeft van alle queries binnen de applicatie, zodat de juiste indexen kunnen worden gelegd.

WebLogic Server ondersteunt zowel lokale als globale transacties. Globale transacties omvatten meerdere resources. Globale transacties hebben extra logging en extra netwerk I/O nodig, wat ze enigszins trager maakt dan lokale transacties. Wanneer mogelijk maak dan gebruik van lokale transacties.

Het gebruik van connectie pools voor databaseconnecties, zijn we het allemaal over eens, daar kunnen

Alleen door uitvoerig te testen kun je achterhalen waar je de oplossing moet zoeken. Zeker voor software applicaties gaat dat op.

we niet meer zonder. Om verzekerd te zijn van een optimale performance, moeten we ervoor zorgen dat er geen requests zijn die staan te wachten op een connectie object uit de pool.

Performanceproblemen oplossen

Nu we de applicatie en de omgeving tot in de perfectie hebben getuned, zijn de gebruikers blij en verwerkt het systeem zonder blikken of blozen honderden requests per seconde, of niet? Zoals al eerder opgemerkt vereist het succesvol oplossen van problemen een goed begrip van het systeem en zijn componenten. Elk systeem is anders en elk performance probleem is waarschijnlijk anders. Er zijn echter een aantal best practices die kunnen helpen bij het lokaliseren van problemen.

Voorbereiding

Het oplossen van problemen is moeilijk en kost veel tijd. Als gebruikers ontevreden zijn, hebben we de juiste infrastructuur, processen en mensen nodig om het probleem aan te pakken.

Ten eerste zou de applicatie uitvoerig getest moeten zijn. Het is belangrijk te achterhalen hoe de applicatie zich in de test gedroeg om te weten of het probleem waar je een oplossing voor moet vinden normaal is tijdens peak loads. Testresultaten brengen ook het resourceverbruik in kaart. Goed testen is het halve werk voor het oplossen van productieproblemen.

Verder moeten alle performance monitoring mechanismen op zijn plaats staan om informatie te geven over de systeemperformance en -activiteit. Helaas zijn performanceproblemen er niet op aanvraag, dus hebben we enige vorm van logging nodig om het resourceverbruik en -activiteit tijdens een bepaalde periode te reconstrueren.

Als laatste hebben we een team en een proces nodig voordat het probleem zich voordoet. Het is geen slecht idee om een multi-disciplinair team te hebben dat verantwoordelijk is voor het oplossen van problemen.

Identificeren en corrigeren van knelpunten

Een knelpunt is een resource binnenin een systeem dat de throughput van een systeem beperkt of significant de responsetijd nadelig beïnvloed. Het fixen van knelpunten in een gedistribueerd systeem is niet een van de makkelijkste taken binnen de IT. In het algemeen zijn er zoals hierboven al opgemerkt multi-disciplinaire teams nodig. Door de komst van performance monitoring tools wordt het fixen van knelpunten wel wat makkelijker. Knelpunten kunnen optreden in de webserver, applicatiecode, applicatieserver, database, netwerk, hardware of operating systemen.
Ervaring leert dat knelpunten meer voorkomen in bepaalde gebieden dan andere, zoals:

- Database connecties en queries
- Applicatie server code
- Applicatie server en web server hardware
- Netwerk en TCP configuratie

Conclusie

Zoals al duidelijk wordt uit de uiteenlopende discussie moet je veel weten en in beschouwing nemen bij het optimaliseren van de performance. Er gaat niets boven ervaring als het gaat om succesvolle performance tuning, en ervaring doe alleen op door het te doen, dus ga de performance tunen. «

Er gaat bij performance tuning niets boven ervaring, dus veel doen is het credo.

Referenties

- R. Patrick, G. Nyberg en P. Aston, "Professional Oracle WebLogic Server", Wiley Publishing, Indianapolis 2010.
- WebLogic Documentatie: http://download.oracle.com/docs/cd/E14571_01/wls.htm