Architectuur van een gedistribueerde muziekspeler

Panradio: zelflerende database

Maarten Fokkinga en Harold van Heerde

Zou het niet makkelijk zijn als er een muziekspeler bestaat die zelfstandig muziek van mijn smaak afspeelt zonder verstoring door reclame, file-informatie en het gepraat van DJ's? Panradio is zo'n systeem.

Panradio biedt de gebruiker muziek aan via een simpele grafische interface. Naast het aan- en uitzetten kan de gebruiker het actuele nummer afbreken en daarmee een nieuwe opstarten. Ook kan de gebruiker een waardering aan het huidige nummer geven, of uit een lijst met voorstellen een nummer 'aanvragen'. Panradio is uitgevoerd als 'vrij project' in de opleiding Technische Informatica door H.J.W. van Heerde (database- en server-aspecten) samen met R.M. Smelik (zelflerende aspecten), onder begeleiding van H.J.A. op den Akker en M.M. Fokkinga.

> Het is zonder experimenten haast onmogelijk te voorspellen wat het effect is van welke maatregelen

Panradio is zelflerend: de speler analyseert het gedrag van de gebruiker bij het afspelen van muziek. Deze muziek herbergt bepaalde kenmerken zoals artiest en genre. De speler houdt bij welke muziek de gebruiker wel of niet mooi vindt, en zoekt nieuwe nummers die overeenkomen met het smaakprofiel van de gebruiker. Panradio is gedistribueerd: de speler kan putten uit een grote collectie muziek en de daarbij behorende informatie die beschikbaar is gesteld op Internet door alle gebruikers.
Van de vele interessante aspecten van Panradio geeft dit artikel een korte schets van de architectuur en gaat daarna dieper in op de systeemprestaties: het aantal gebruikers dat het systeem aan kan, en de wijze waarop de prestaties geanalyseerd en verbeterd kunnen worden. Meer informatie over Panradio is op Internet (http://panradio.vanheerde.net/panradioverslag.pdf) te vinden.

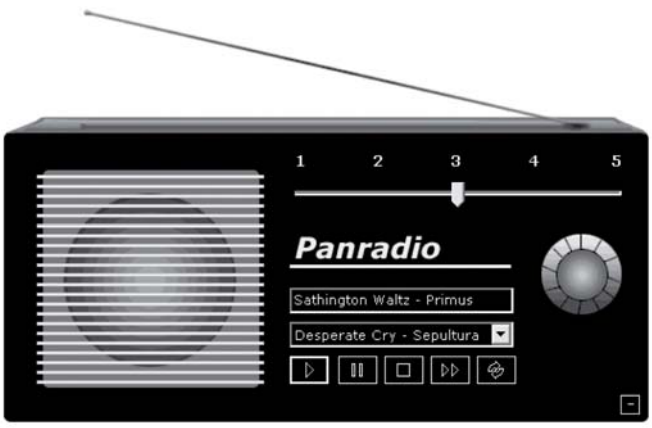
Architectuur

Om verwarring te voorkomen gebruiken we vanaf nu het woord song in plaats van nummer. Panradio is een gedistribueerd systeem bestaande uit een of meer servers en per gebruiker één client, zie afbeelding 2. Een client is opgebouwd uit drie componenten:

- Agent, deze zorgt voor het kiezen van songs gebaseerd op een *smaakmodel* van de gebruiker;
- FileManager, zorgt voor het downloaden uit andere clients van de songs die gekozen worden door de Agent. Daarnaast vergaart de FileManager informatie over de lokale muziekverzameling van de gebruiker. (Dit gebeurt in het prototype aan de hand van de ID3v1-tag van MP3's; zie www.id3.org/id3v1.html);
- Player, speelt de gedownloade bestanden af, en geeft een interface voor de feedback van de gebruiker.

De server beheert in een relationele database de informatie over de totale muziekcollectie. We spreken gemakshalve over een 'database van songs', terwijl in feite in de database alleen informatie *over* songs is opgeslagen; de songs zelf staan bij de gebruikers op de computer. De server bestaat uit twee componenten:

- InfoManager, deze accepteert *SongQuery's* van de Agents en retourneert de gevraagde songs uit de database. Een SongQuery specificeert de gewenste en ongewenste kenmerken van songs.
- NameServer, deze zorgt ervoor dat de songs in de database terecht komen. Daarnaast kunnen FileManagers aan de NameServer locaties van songs opvragen.



Afbeelding 1: Panradio gebruikersinterface.

Om de programmering te vergemakkelijken is alle SQL ingekapseld in functies zoals getTitle en getPerformer. Helaas leidt dat tot enige inefficiëntie, bijvoorbeeld wat soms met één SQL query kan (zoals de title én name selecteren uit een join van tabellen song, performs en performer) leidt door gebruik van die functies tot twéé SQL query's. Om toch tot acceptabele query-tijden te komen is het ontwerp uitgebreid geëvalueerd door middel van tests en zijn aan de hand daarvan indexen gespecificeerd.

Door een toename van het aantal gebruikers zal de database groeien, met als gevolg dat executietijden ook toenemen tot uiteindelijk een onacceptabele duur. Om te voorspellen hoeveel gebruikers een server aankan met het huidig ontwerp, is er een model nodig van de prestaties van de server. Uit zo'n model kunnen ook verbeteringen voor het ontwerp afgeleid worden, waardoor er méér gebruikers mogelijk zijn.

> Optimalisaties kunnen worden bereikt door het aanleggen van indices

We laten nu zien hoe zo'n model is opgesteld, door de prestaties van het Panradio-prototype, met één server, te meten in een aantal tests en hoe dat model vervolgens gebruikt is. De tests meten de executietijden van het indexeren van nieuwe songs, het opvragen van songs en het opsporen van locaties van songs, als functie van de database-grootte. De tests zijn niet alleen nodig voor het opstellen van een model, maar ook om te beslissen welke maatregelen (zoals het aanleggen van indices op diverse attributen) daadwerkelijke optimalisaties zijn en welke dat niet zijn. Het is zonder experimenten haast onmogelijk te voorspellen wat het effect is van welke maatregelen.

Tests

Alle tests zijn uitgevoerd op een database draaiend op een server met een AMD Athlon XP 2500+ processor, een 80 GB 7200 RPM Samsung HD, 256 MB DDR geheugen en een Ashrock moederbord. Het DBMS is Postgresql 7.4. De tests zijn zoveel mogelijk uitgevoerd op tijden dat de server niet belast wordt door andere processen. Bij de tests worden er kunstmatige, niet-bestaande Songs, Performers, Genres, Albums en Years gegenereerd. Hierbij wordt ervoor gezorgd dat er per Performer twintig Songs zijn. Elke Song krijgt één locatie.
Om het indexeren van songs te modelleren zijn er series van 100 songs gegenereerd en naar de server gestuurd. Optimalisaties kunnen worden bereikt door het aanleggen van indices (btrees). De executietijd van indexatie i, gemeten in milliseconden, hangt lineair af van het aantal songs n in de database:
$i = 0.0014*n + 64$
Optimalisatie verkort de executietijden drastisch. Het DBMS probeert zelf een goede query-executiestrategie te bedenken en past deze aan als een bepaalde grens (100.000 tuples) is bereikt. Voor het bepalen van de gemiddelde executietijd van het opvragen van een song, worden er verschillende, voor het systeem representatieve, SongQuery's gegenereerd en naar de InfoManager gestuurd. De gemiddelde query-tijd q hangt als volgt af van het aantal songs n in de database:
$q = 0.0066*n + 30$
Indien er geen optimalisaties worden toegepast, is bij 120.000 songs in de database de executietijd zonder indices ongeveer 20 seconden, mét indices minder dan 1 seconde. Bij een grotere waarde van n wordt de verhouding snel groter: bij 500.000 songs is de executietijd zonder indices ongeveer 4.5 minuut, mét indices slechts 1.5 seconde.
De test voor het vinden van de functie voor de executietijd van een locatie-opvraag gaat analoog aan de hiervoor beschreven tests. Er wordt een vraag naar een locatie van een willekeurig gekozen song naar de NameServer gestuurd. De executietijd l van een locatie-opvraag blijkt als volgt af te hangen van het aantal songs n in de database:
$l = 0.0023*n + 48$

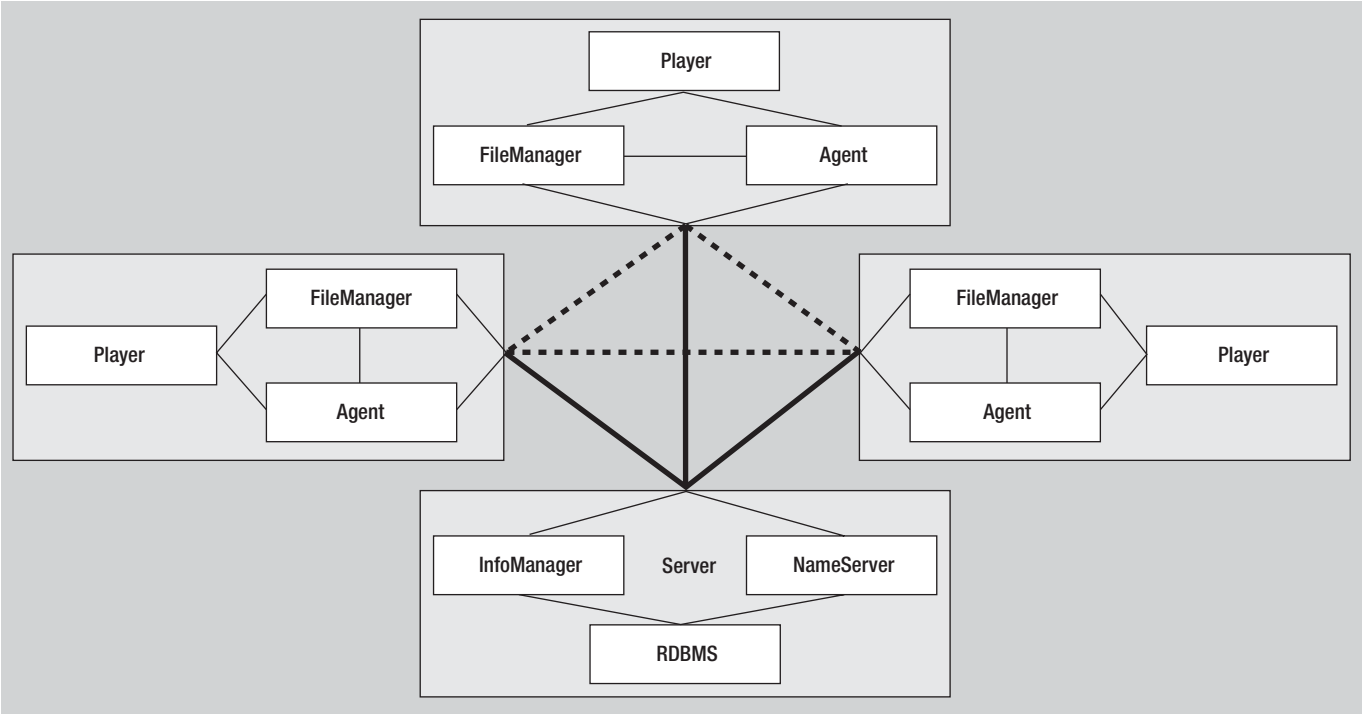
Model

Uit de tests is gebleken dat met name geschikt gekozen indices de executietijden aanzienlijk verbeteren. Daarbij zijn er goede benaderingen gevonden van de verschillende executietijden als functie van het aantal songs in de database. Het doel is nu om het gedrag van de totale server (InfoManager + NameServer) te modelleren.
Naast de al genoemde n, i, q en l, worden ook de volgende variabelen onderscheiden:

g: het aantal gebruikers van Panradio;
a: het aantal song-aanvragen per tijdseenheid;
v: de totale verwerkingstijd van een aanvraag;
p: de prestatie, i.e., het aantal verwerkte songs per tijdseenheid: $p = 1/v$;
A: de fractie van de gebruikers die actief zijn (alleen zij veroorzaken aanvragen van songs);
S: de gemiddelde tijdsduur van een song;
F: een factor in tussen 0 en 1 die van de Agents afhangt;
R: de verhouding (ratio) tussen aantallen SongQuery's en locatie-opvragingen.

De met een kleine letter geschreven variabelen hangen af van het aantal gebruikers, g. De variabelen zijn als volgt aan elkaar gerelateerd (met milliseconde als tijdseenheid):

- We nemen aan dat elke gebruiker 1000 verschillende songs heeft, en dat deze verschillen van de songs van de andere gebruikers. Dus voor het aantal songs in de database geldt: $n=1000*g$.
  Deze aanname is discutabel: het is de vraag of bij grote gebruikersaantallen iedere nieuwe gebruiker wéér 1000 nieuwe



Afbeelding 2: Architectuur van Panradio. De lijnen en stippellijnen tussen de componenten duiden communicatie aan.

songs inbrengt. Maar met het verstrijken van de tijd zullen er wél steeds nieuwe songs op de markt komen en door gebruikers ingebracht worden. Bij gebrek aan beter houden we het op de gedane aanname.

- Het aantal song-aanvragen per tijdseenheid a, is rechtevenredig met het aantal actieve gebruikers A*g en omgekeerd evenredig met de tijd F*S die het afspelen van een song duurt; we stellen dat gemiddeld genomen een gebruiker een song afbreekt nadat fractie F ervan is afgespeeld. Dus a=A*g/(F*S). Betere Agents zorgen ervoor dat de gespeelde songs meer in de smaak van de gebruiker vallen en de gebruiker dus minder songs afbreekt, zodat factor F dichter bij 1 ligt.
- De totale verwerkingstijd van een aanvraag, v, is per definitie de som van de verwerkingstijden van de InfoManager en de verwerkingstijden van de NameServer voor het traject tussen het zoeken naar een geschikte song en het leveren van een locatie voor de song. We nemen ter vereenvoudiging aan dat het zoeken van songs (SongQuery's), het opvragen van locaties en de indexaties niet tegelijkertijd worden gedaan. In werkelijkheid gaat dit deels wel tegelijkertijd. We nemen voorts aan dat na iedere SongQuery en iedere locatie-opvraag een indexatie plaats vindt (zodat er volgens deze modellering altijd "tijd genoeg is voor indexaties"). Omdat er per locatie-opvraag R SongQuery's zijn, is de totale verwerkingstijd van een song-aanvraag, v, gelijk aan: R*(q+i) (voor de verwerking van de SongQuery's plus opvolgende indexaties) plus l+i (voor de locatie-opvraag en de erop volgende indexatie).

Samengevat leveren deze overwegingen en aannamen, tezamen met de tests, het volgende model op:

i = 0.0014*n + 64
q = 0.0066*n + 30
l = 0.0023*n + 48
n = 1000*g
v = R*q + l + (1+R)*i
a = A*g / (F*S)
p = 1/v

Uiteraard kan het model verfijnd worden met meer aspecten en gedetailleerder aannamen.

Gebruik van het model

Een belangrijke prestatie-eis aan Panradio is dat een gebruiker nooit hoeft te wachten op een volgende song. Dit betekent dat een aanvraag uitgevoerd moet zijn binnen de tijd dat de voorafgaande song afgespeeld is. In termen van het model is hieraan *gemiddeld genomen* voldaan wanneer:

p >= a

Bij gebruikersaantallen tot en met 292 kan er 20 procent (ongeveer 58), tegelijkertijd actief zijn, zonder dat een van hen moet wachten wanneer een nieuwe song afgespeeld gaat worden.

Het model geeft aan dat de volgende aspecten van grote invloed zijn op de schaalbaarheid van het Panradio-systeem.

A. De richtingscoëfficiënten van i, q en l geven aan hoe groot de toename in tijd is bij een toename in het aantal songs in de database. Hoe kleiner deze coëfficiënten, hoe kleiner die toenamen en hoe korter de executietijd bij grotere aantallen songs in de database. Een verkleining van de coëfficiënten heeft tot gevolg dat er meer gebruikers van het systeem gebruik kunnen maken.

Bijvoorbeeld, het einde van Sectie beschrijft hoe query's worden gegenereerd. De manier waarop dat gebeurt bepaalt mede de coëfficiënt van q.

B. Indien het aantal songs in de database lager is, dan is de prestatie hoger. Dit suggereert om de database te distribueren over verscheidene servers.
C. De verhouding R tussen aantallen SongQuery's en locatie-opvragingen speelt een belangrijke factor in het aantal gebruikers. Hoe meer SongQuery's er per songaanvraag uitgevoerd moeten worden, hoe groter de belasting.
D. Het percentage A van gebruikers dat actief is, is van grote invloed op het aantal gebruikers. Door verschillende waarden voor A te kiezen (A*g is het aantal actieve gebruikers) kunnen verschillende gebruikersscenario's doorgerekend worden.

Om de programmering te vergemakkelijken is alle SQL ingekapseld in functies

Punten A en B beïnvloeden de prestatie aan de server-zijde. Punt C is, net als factor F, afhankelijk van de Agents: indien de Agents zorgvuldig omgaan met resultaten zijn er minder query's nodig, wat gunstig is voor de belasting van het systeem. Punt D is een statistische waarde die moeilijk beïnvloed kan worden.

Tot slot

Om het database-ontwerp van het prototype van Panradio te evalueren, en enig houvast te hebben bij verbeteringen aan het ontwerp, is er een model van het ontwerp gewenst. Zo'n model is gemaakt; het past op de achterkant van een bierviltje. De methode kan in principe toegepast worden op elk ontwerp om op een relatief eenvoudige manier het ontwerp te evalueren en te verbeteren. Om goede benaderingen te vinden voor sommige executietijden die in het model voorkomen, zijn uitgebreide tests gedaan. Die tests zijn in ieder geval ook nodig om te bepalen welke maatregelen de prestaties verbeteren. Het aantal gebruikers is dé maatstaf waaraan de prestatie van het systeem afgemeten wordt. Er is inmiddels een ontwerp gemaakt voor de distributie van de data over meerdere databases. Verder onderzoek zal moeten uitwijzen of het ontwerp schaalbaar is, zodat niet alleen onze huisgenoten, of de vele campusleden, maar ook enorme aantallen Internet-gebruikers kunnen luisteren naar Panradio!

Maarten Fokkinga en **Harold van Heerde**
M.M. Fokkinga (fokkinga@cs.utwente.nl) en H.J.W. van Heerde (h.j.w.vanheerde@student.utwente.nl) zijn werkzaam op de Universiteit Twente, Faculteit EWI.