Data synchronisatie in online multiplayergames

Joshua Moerman Nick Overdijk Rik Harink

15 juli 2010

Inhoudsopgave

1 Inleiding

"Most people would succeed in small things if they were not troubled with great ambitions." - Henry Wadsworth Longfellow. Ambitie is een mooi iets, het is de drijfveer om grootse dingen te doen en prachtige dingen te maken. Soms echter kunnen te grote ambities je voortgang echter tegen gaan. Goed is niet goed genoeg en je product moet op zijn minst wereldschokkend zijn.

Ons grootste struikelblok bij het ontwikkelen van ons framework en dit gebruiken voor een onderzoek naar synchronisatie was onze eigen ambitie. We wilden een perfect framework ontwikkelen dat razendsnel kon communiceren over het internet en daarmee multiplayer-spellen kon ondersteunen voor duizend spelers. Dit alles wilden wij ontwikkelen met z'n drieën in een halfjaar tijd.

Uiteindelijk hebben we een framework gebouwd dat daadwerkelijk kan synchroniseren. Onze doelstelling om 1000 spelers te kunnen ondersteunen doormiddel van revolutionaire concepten hebben we echter in de loop van de tijd laten varen, dit was simpelweg niet realiseerbaar in deze korte tijd. We kunnen echter al wel meer dan twintig spelers ons spel laten spelen zonder haperingen. Dit alles gebeurd met een ontzettend simpel model dat makkelijk te implementeren is in games. Al met al hebben we nu een redelijk robuuste fundering ontwikkeld voor een open-source synchronisatie platform. Deze fundering heeft zeker potentie om uit te groeien tot een compleet product geschikt voor echte games.

Bij dit framework hebben wij een representatie test game te zoeken om te onderzoeken of ons framework voldoet aan onze eisen.

1. Ondersteuning voor 20 spelers of meer
2. Latency $\leq$ 180 ms [Arm03]
3. Synchroniciteit is groot genoeg voor een vloeiende speelervaring

2 Theoretisch kader

2.1 Probleemstelling

- *Is event-based synchronisatie voldoende om een multiplayer-game te synchroniseren zodanig dat de netwerkvertraging niet storend is?*

De term ëvent-based synchronisatie"wordt later in dit stuk uitgelegd, net zoals wanneer de netwerkvertraging storend wordt.

Ons onderzoek concentreert zich met name op welke manieren men data kan synchroniseren. Om dit te onderzoeken hebben we een framework geschreven waarin wij helemaal vrij zijn om op verschillende manieren te synchroniseren. Om data te hebben om te synchroniseren hebben we ook een eigen game geschreven.

Wij zien een optimalisatie als een aanpassing die de netwerkvertraging beter (dan andere methodes) verbergt of daadwerkelijk terugdringt. Om te weten waarop wij kunnen optimaliseren hebben wij het proces van synchroniseren geanalyseerd.

2.1.1 Synchronisatie analyse

Als we het server-cliënt model aannemen, wat later uitgelegd zal worden, kunnen wij synchroniseren in het algemeen in de volgende stappen beschrijven:

1. Cliënt A stuurt zijn staat naar server.
2. De server ontvangt zijn bericht en stuurt het door naar alle andere cliënten
3. De andere cliënten ontvangen de nieuwe staat van de van de server en pakken het uit
4. De andere cliënten laten de staat van de ene cliënt zien

Hierbij zijn wij tot de conclusie gekomen dat de bottleneck de bandbreedte van verbindingen is waardoor wij vooral zijn gaan zoeken naar optimalisaties die de hoeveelheid data die verzonden moet worden beperkt. Hoe minder data nodig is om synchroon te blijven, hoe meer cliënten er synchroon kunnen blijven voordat de verbinding verstopt raakt en netwerkvertraging opstapelt. De optimalisaties kunnen optimaliseren op de volgende variabelen winst boeken.

- De frequentie waarmee een cliënt zijn staat verstuurt
  Als de cliënt zijn staat minder vaak verstuurt scheelt dat in het verbruik van de bandbreedte.
- De data die een cliënt verstuurt
  Hoe minder data de cliënt hoeft te versturen/ontvangen om weer synchroon te zijn, hoe minder de bandbreedte wordt gebruikt.

Wij hadden een aantal ideeën hoe de synchronisatie verbeterd zou kunnen worden.

Als er in onderstaande tekst wordt gesproken over "De staat"dan wordt de staat van het programma wat gesynchroniseerd dient te worden bedoeld.

- Ongeoptimaliseerd
  - De state wordt elk iteratie verstuurd. Ongeacht of er iets is veranderd in de huidige cliënt wordt toch zijn staat verstuurd. Dit is een van de simpelste manieren van synchroniseren maar is zoals verwacht ook erg inefficiënt en zelfs bij een klein aantal cliënten al te belastend.

- Out-of-Sync detectie
  Out-of-Sync detectie is belangrijk om de efficiëntie van synchronisatie te verhogen. Immers hoeft er alleen data verstuurd te worden als er een client out-of-sync is. Out-of-Sync detectie is echter lastig om zelf efficiënt te doen: hoe kan de server weten dat een cliënt out-of-sync is zonder dat hij de data van die cliënt weet?
  Over dit probleem hebben wij in eerste instantie zelf nagedacht. Het idee kwam er om de cliënt en server bepaalde voorspellings-algoritmes te laten delen. Met deze voorspellingsfunctie is in principe de hele data te voorspellen vanaf een willekeurig punt. De server weet deze algoritmes ook, en weet ook in welke staat de cliënt zich nu bevindt. Omdat de server dit weet kan de server, zonder extra data te ontvangen van de cliënt, weten of een cliënt out-of-sync is.
  - Voorspellen
    * Als de data die gesynchroniseerd dient te worden voorspelbaar is, is hiervan goed gebruik te maken voor het detecteren van out-of-sync, zoals hierboven is uitgelegd. Ervan uitgaande dat de server weet wat de cliënt op het moment denkt weet de server of hij de data moet verzenden. Bij goede voorspellingsfuncties kan dit een grote winst opleveren, zowel voor een server-cliënt als een p2p model. Voorspelfuncties vereisen wel dat de server domeinkennis heeft, waardoor de *generiekheid* van de server verloren gaat. Merk ook op dat, in de huidige versie van onze game, ook een voorspelfunctie zit, maar deze functie er alleen is om de speler van die cliënt het idee te geven dat zijn cliënt nog goed in sync is, de server verstuurt alle data zelfs als de ontvangende cliënt de data 100% goed heeft voorspeld. De DARPA heeft een tijd geleden, voor 1997 [Aro97], een systeem ontworpen wat erg op de voorspelling lijkt. Het systeem heet *Dead Reckoning* en houdt in dat zolang een object A niet genoeg afwijkt bij een client, dat object niet wordt gesynchroniseerd. Dit lijkt op de voorspelfunctie in dat er data wordt verstuurd dan en slechts dan als de client out-of-sync is. Wat onze voorspelfuncties echter zouden kunnen doen is spelergedrag simuleren zodat de vertraging echt verborgen wordt en de data-integriteit 100% lijkt voor de gebruiker.

        * De voorspelfuncties moeten door de programmeur zelf worden geschreven. Hier hebben wij voor gekozen zodat het framework klein en licht blijft, en omdat de algemene voorspelfuncties domeinkennis misschien waardoor ze nooit zo goed kunnen zijn als wat de programmeur zelf kan produceren. Generieke voorspelfuncties zouden idealiter wel in het framework zitten zodat de programmeur niet wordt verplicht zijn eigen implementatie te geven, maar de keuze heeft om dit te doen.
    - Event-Based optimalisatie
        * Bij deze vorm van optimalisatie wordt de state verstuurd zodra er een verandering heeft plaatsgevonden die dusdanige groot is dat de andere cliënten een update nodig hebben om hun voorspelling te laten kloppen. Bij onze demo-game is dit als de lokale speler van richting verandert: zolang de speler niet van richting verandert voorspellen de andere cliënten de positie van de speler correct.

Bij beide vormen synchronisatie meten wij de vertraging tussen de cliënten. De acceptabele vertraging voor onze game is 180ms [Arm03]. Meer dan 180ms zal rekenen wij als storende netwerkvertraging.

2.2 Communicatie

Voor het delen van data tussen verschillende computers zijn verschillende modellen mogelijk, wij noemen er hier 3 waarvan wij verwachten en weten [CKFJ04] dat ze het meeste voorkomen uit persoonlijke ervaring met multiplayer games over het internet.

- Server to Cliënt
  Bij dit model verbindt elke cliënt met dezelfde server, die vervolgens ervoor zorgt dat de updates van cliënten doorgestuurd wordt naar de andere cliënten, mocht dit nodig zijn.
    - Voordelen
        * Simpel op te zetten
        * Server is betrouwbare bron van informatie, kan dienen als punt van verificatie van de data
    - Nadelen
        * De server zijn verbinding wordt snel overbelast. Als er 1 cliënt bijkomt moet de server de data van alle anderen cliënten ook naar die cliënt sturen, en naar alle andere cliënten ook de data van de nieuwe cliënt. Hierdoor stijgt het bandbreedte gebruik erg snel.
- Peer-To-Peer
  Bij dit model verbinden de cliënten onderling. Elke client is efficiënt een kleine server

die zijn updates naar elke andere cliënt stuurt en zodoende van de andere cliënten de data ontvangt.

  - Voordelen
    * De bandbreedte verdeling is eerlijk en optimaal, wat de vertraging ten goede komt.
  - Nadelen
    * Per cliënt vereist het wel meer van de verbinding, maar in de praktijk nooit onredelijk veel.
    * Last van connectiviteitsproblemen omdat een directe connectie met IPv4 lastig gaat door routers heen.

- Hybrid
  Een model waarbij computers onderling proberen te praten en mocht dit niet lukken de server een handje kan helpen. Ook kan de server dienen als dataverificatiepunt, wat bij games neerkomt op cheat-detectie.

  - Voordelen
    * Probeert het beste van beide te combineren.
    * Connectiviteitsnadelen van P2P, worden deels opgevangen door de server. Als er totaal geen directe connecties mogelijk zijn wordt dit wel een probleem, dan verandert dit model van het hybride model naar het server model.
  - Nadelen
    * Lastiger op te zetten in software.

In ons framework maken wij gebruik van de cliënt-server vorm. Hier is voor gekozen omdat dit model veel gebruikt wordt in de gamewereld[CKFJ04] en ons het makkelijkst leek te implementeren. Achteraf gezien denken wij er nog hetzelfde over.

2.3 Synchronisatie

Als we het server-cliënt model aannemen kunnen wij synchroniseren in het algemeen in de volgende stappen beschrijven:

1. Cliënt A stuurt zijn staat naar server.
2. De server ontvangt zijn bericht en stuurt het door naar alle andere cliënten
3. De andere cliënten ontvangen de nieuwe staat van de van de server en pakken het uit
4. De andere cliënten laten de staat van de ene cliënt zien

Met deze eenvoudige stappen kunnen er weinig gevariëerd worden. De enige variabelen die we hebben zijn:

- De momenten waarop een cliënt iets verstuurt
- De data die een cliënt verstuurt

De event-based variant van ons synchronisatie-framework is gebaseerd op punt 1, de momenten waarop een cliënt iets verstuurt, hij verstuurd alleen de data van de cliënt als er een event heeft plaats gevonden, in ons geval het veranderen van richting bij cliënt A. De data die verstuurd word kan ook geminimaliseerd worden, je hoeft bijvoorbeeld niet alle data die je van de andere cliënten hebt te sturen naar de server. Deze data heeft de server namelijk al, je hebt de data immers in de eerste plaats van de server ontvangen. Alleen het sturen van jouw eigen data is voldoende.

3 Hypothese Toetsing

3.1 Het meten van de data-integriteit

3.1.1 Methode

Elke client logt z'n gamestate vlak voor het synchroniseren met de andere clienten en de tijd waarop hij dat doet. De tijd loopt synchroon tussen alle clienten. Door de logs te vergelijken kunnen we zien hoe ver de clienten out of sync waren op dat tijdstip door gamestates te vergelijken. We maken hierbij de aanname dat als 2 gamestates binnen een marge van 100ms erg op elkaar lijken, dit dezelfde gamestate was. De parser gaat voor alle gamestates, voor elke gamestate in de andere client binnen 100ms, welke het meest op hem lijkt. Degene die het meest op hem lijkt wordt als dezelfde beschouwd, het verschil in gamestate genoteerd en de timestamps van elkaar afgetrokken voor de vertraging **To do (??)** .

3.1.2 Opstelling

Als opstelling hadden we 25 pc's klaargezet met Fedora Linux in 1 van de TK's. Op een aantal pc's werden de clienten tegelijkertijd gestart en verbonden deze met de server op een MacBook Pro 5,5 op hetzelfde netwerk. Wat voor netwerk-apparatuur er tussen de pc's lag is onduidelijk: het was het netwerk van de Radboud, terminal kamer 00.71 **To do (??)**

Door het aantal cliënten te verhogen per meting kan worden gekeken hoe het aantal clienten van invloed is op de vertraging tussen clienten. Door het verhogen van het aantal cliënten moet er meer data gesynchroniseerd worden waardoor de server het drukker krijgt.

4 Resultaten

De meetresultaten laten ons zien dat onze game niet storend hoeveelheid netwerkvertraging had. Onze gemiddelde netwerkvertraging was 47ms en de gemiddelde afwijking was 13pixels. Dit is allemaal ruim binnen acceptabele marges [Arm03]. Voor de gehele resultaten zie de bijlage. In de resultaten hebben wij gemeten over een aantal cliëten van het BunnyBounce spelletje, onze testgame, hoeveel netwerkvertraging er was bij synchronisatie met behulp van ons framework JNRSync met event-based-synchronisatie op het netwerk en de computers van de universiteit.

5 Discussie

De metingen in ons framework gaan uit van het feit dat alle gebruikte computers dezelfde lokale tijd hebben op de milliseconde nauwkeurig. Verder worden de verkrijgte resultaten vergeleken met andere resultaten die binnen een bepaalde marge van de gedane meting lagen. Als laatste probleem met de manier van meten is dat de computers zoveel logbestanden aanmaakten dat de pc's waarop de cliënten draaiden overduidelijk vertraging opliepen, dit kan onze meetresultaten hebben verstoord.

Potentieel kan ons framework meer spelers dan 20 aan. Dit is echter materiaal voor een vervolg onderzoek omdat wij niet genoeg computers tot onze beschikking hadden om meer dan 20 cliënten te testen. We hebben nu niet de grens van ons framework opgezocht met 20 cliënten was er nog geen sprake van zichtbare ernstige vertraging van de speelervaring. Meerdere cliënten op dezelfde pc draaien was geen optie omdat er dan concurrency problemen gaan opspelen waardoor de metingen onjuiste data opleveren.

Als vervolg onderzoek zouden wij onszelfd de volgende vragen stellen:

- Wat is de bovengrens qua aantalspelers met een event-based synchronisatieframework zonder verdere optimalisaties?
- Welke optimalisaties kunnen toegepast worden in het synchroniseren van online games?

6 Conclusie

De resultaten geven aan dat ons framework goed speelbaar is met 20 cliënten. Onze maximale bovengrens was 180 ms latency en ons framework blijft daar ruim onder met een op zijn hoogst gemeten 70 ms en een gemiddelde latency van 50 ms. Ons framework is dus geschikt voor multiplayer games met 20 spelers. Verder valt ook te zien dat het toenemen van het aantal spelers geen drastisch effect heeft op het toenemen van de latency.

7 Reflectie

7.1 Groepswerk

De samenwerking in onze groep is uitermate soepel verlopen. Het werk is goed verdeeld en iedereen was uitermate enthausiast om zijn steentje bij te dragen. Regelmatig hebben we avonden of middagen geplanned om met zijn drieën te werken aan en te brainstormen over het framework en het verslag, dit beviel erg goed omdat je zo direct feedback kon geven op elkaars werk.

Door gebruik van het versiecontrole systeem git was het verder ook uitermate makkelijk om je gedane werk te delen met je groepsgenoten. Verder voorkomt het het kwijtraken van code en houd je ook automatisch een soort logboek bij.

7.2 Verbeterpunten

Verbeterpunten waren er natuurlijk wel aanwezig. We hadden beter in de pilot fase al wat reëlere doelen kunnen stellen. Dit had een hoop stress en tijd gescheelt. Dit omdat we nu redelijkt wat tijd kwijt zijn geraakt aan onderdelen van het framework die we uiteindelijk niet af hebben kunnen krijgen of niet werkend hebben kunnen krijgen.

Verder hadden we achteraf gezien eerder moeten beginnen met het onderzoekdeel, met het developmentdeel zat het helemaal goed maar het onderzoek had eerder voorbereid en gedaan moeten worden. Nu kwam alles op het laatste moment aan en toen vielen opeens zowel Joshua als Nick weg wegens omstandigheden. Hierdoor was de eerste versie van het uiteindelijke onderzoeksverslag onder de maat. Eerder beginnen aan het onderzoek had een hoop stress gescheelt en het had meer tijd gecreëerd voor het maken van het verslag.

A Begrippen

sed Synchronisatie	Het principe dat je alleen je eigen data synchroniseert als er een event heeft plaatsgevonden.
Multiplayer Game	Een spel waar meerdere mensen aan deel kunnen nemen. Vaak gaat de data van dit spel over het internet.
Framework	Een verzameling code die gebruikt kan worden door anderen waardoor zij de functionaliteit van desbetreffend framework niet opnieuw hoeven te implementeren.
JNRSync	Het framework wat wij hebben geschreven wat ons onderzoek gemakkelijker maakte.
Synchroon	Twee verschillende dingen beschikken over dezelfde informatie op hetzelfde moment
Netwerkvertraging	de tijd die tussen het daadwerkelijk gebeuren van een gebeurtenis en het ontvangen van deze gebeurtenis door een andere client zit.
Peer-To-Peer	Een setup waarin cliënten onderling verbinden via een netwerk.
Client-Server	Een setup waarin cliënten met een server verbinden via een netwerk.
Mirrored-Server	Een server die alle data die hij ontvangt doorstuurt aan de verbonden cliënten.
IPv4/IPv6	Protocollen waarin is afgesproken hoe apparaten op een netwerk kunnen worden geadresseerd. Het verschil tussen IPv4 en IPv6 is dat IPv6 2^16 meer adressen heeft dan IPv4 waardoor elk apparaat een uniek adres kan krijgen.

Referenties

[Arm03] G. Armitage. An experimental estimation of latency sensitivity in multiplayer Quake 3. In *Proceedings of the 11th IEEE International Conference on Networks (ICON 2003)*, pages 137–141, 2003.

[Aro97] Jesse Aronson. Dead reckoning: Latency hiding for networked games. 1997.

[CKFJ04] ERIC CRONIN, ANTHONY R. KURC, BURTON FILSTRUP, and SUGIH JAMIN. An efficient synchronization mechanism for mirrored game architectures. 2004.