thema

Het ontwikkelen van een besturingssysteem zou je kunnen beschouwen als de hoge school van softwareontwikkeling. Het is ook werk dat zowel voor als door de ontwikkelaar gedaan wordt. In het geval van de ontwikkeling van DTrace is dat wel in zeer hoge mate het geval. Uit onderstaand gesprek met DTrace-ontwikkelaar Bryan Cantrill blijkt bovendien dat er een hele filosofie achter het werken aan een besturingssysteem zit. *En passant* vertelt Cantrill ons wat er nog moet veranderen aan software-ontwikkeling in het algemeen.

'Het maken van een besturingssysteem zit in je bloed'

Requirements spelen een ondergeschikte rol

Ter gelegenheid van de introductie van Solaris gaf het complete driekoppige team dat DTrace ontwikkeld heeft, een kleine presentatie. Eén ontwikkelaar viel op, door zijn pregnante manier van praten. ('Er is helaas nog steeds geen bit voor software die door sukkels geschreven is.') Hij gaf ook een aantal tot de verbeelding sprekende voorbeelden van fouten in software, die eigenlijk alleen door de toepassing van DTrace gevonden hadden kunnen worden.

Cantrill: 'Ik denk dat het probleem is, dat we software niet kunnen zien, het heeft geen fysieke manifestatie. Ik heb me vaak afgevraagd: hoe zou het zijn wanneer je echt een kamer binnen zou kunnen gaan en in die kamer zou het besturingssysteem zijn, of het software systeem. We zouden tandwielen en pompen zien, en allerlei stomme dingen die we deden. 'De memory allocation machine klinkt alsof hij gaat ontploffen, en wat is hier aan de hand!', zoiets. Omdat we software niet kunnen zien, spenderen we veel tijd aan software, om er zeker van te zijn dat het werkt. Maar we begrijpen niet noodzakelijkerwijs de efficiëntie van deze pogingen, omdat de tools daarvoor nooit bestonden. De tools om software te ontwikkelen zijn in wezen niet veranderd in twintig jaar. Steeds wanneer we ontwikkelden voor een nieuwe omgeving, Java, Perl Python et cetera, zijn we gewoon teruggegaan en hebben we de bestaande toolmethodologieën opnieuw geïmplementeerd voor de nieuwe omgevingen. Ontwikkelaars zitten nog steeds met de tools die ze altijd gehad hebben, breakpoint-stijl debuggers, developmenttools die zich focussen op development, niet op wat er in productiesystemen gebeurt. Wij geloven dat de belangrijkste problemen optreden in productiesystemen, niet in developmentsystemen. We moeten naar een model toe waar onze tools die software begrijpen net zo goed in de productiefase ingezet kunnen worden als tijdens de ontwikkelfase. Omdat dat soort tools niet bestaan hebben, vonden wij al die vreemde dingen (*zie bijvoorbeeld http://www.sun.com/bigadmin/content/dtrace/dtrace_usenix.pdf, red.*). Een voorbeeld: in veel IT-omgevingen zijn commando's verpakt in iets dat aangeeft wie het programma draait, om redenen van licenties en zo. Deze wrapper werkt zo, dat hij berekent hoe lang zo'n programma gebruikt wordt. Het wacht min of meer op de achtergrond terwijl je het programma gebruikt. Wanneer het programma stopt, dan berekent het de looptijd. Om voor mij onduidelijke redenen, is dat programma met een wijziging uitgebreid. Ieder half uur zou het actief worden, om te controleren of het programma nog liep. Het deed dat door een lijst op te vragen van alle processen die in het systeem lopen. Niet echt elegant, maar wat doet het ertoe. Het grappige is dat je op die grote Sun-Ray servers met duizenden processen te maken hebt, en op dit specifieke systeem kregen we iedere elf seconden een process listing. Voortdurend was er wel een programma dat riep: "O, mijn dertig minuten zijn om, ik wil een proceslijst." De persoon die het oorspronkelijk schreef, had geen idee dat het tot die schaal uitgebreid zou worden. Met DTrace konden we heel snel precies zien wat er gebeurde en het probleem was dan ook snel opgelost. Zonder DTrace hadden we het waarschijnlijk niet kunnen oplossen. Omdat deze processen zo kort leven, duiken ze niet eens op in de telling, maar toch consumeerden ze behoorlijke hoeveelheid van de CPU-resources. Wanneer je uitgaat van de symptomen, is het heel moeilijk om de reden te ontdekken van de CPU-belasting van dat specifieke systeem. Het is ook symptomatisch voor veel van de problemen die we zien: we namen bijvoorbeeld een wrapper-script, en we drukten het uit het design-centrum, uit het gebied waar het eigenlijk voor geschreven was. Het was geschreven voor een desktop met zeven processen, en nu draaide het met duizenden processen op een grote server. Ik denk dat zoiets steeds vaker gebeurt. Wat je ziet is: een ontwikkelaar ontwerpt bijvoorbeeld een mailparser, die werkt goed op zijn desktop en hij maakt er open source van. Hij kan niet bevroeden dat een groot bedrijf zijn code oppakt en het in een grote mailserver gebruikt en ineens wordt het een groot performance probleem. Je kunt het de ontwikkelaar niet kwalijk nemen, want je drukt het uit het designcentrum. Je moet een manier hebben om het probleem te begrijpen, in productie. Het is een combinatie van een gebrek aan tools, en het feit dat dingen steeds vaker buiten hun oorspronkelijke context gebruikt worden, waardoor we al die problemen met behulp van DTrace gevonden hebben.'

> 'We hebben ontdekt dat sommige aannamen met betrekking tot het systeem gewoon niet kloppen'

VEEL GELEERD *Is DTrace ook van nut geweest tijdens de ontwikkeling van Solaris 10?*

Cantrill: 'Nou en of! Ik heb het gevoel dat iedere keer dat ik DTrace aanzet, ik iets over Solaris leer.'

Solaris heeft een lange geschiedenis, dus de kans dat dingen buiten hun context gebruikt worden, is dan ook heel groot, of niet?

Cantrill: 'Ja, het design van Solaris begon in 1969. Er is code bij die teruggaat tot het oorspronkelijke System 7 Unix.'

En er zal ook wel niet veel documentatie zijn?

Cantrill: 'Nou, dat is gemixt. Zeker met betrekking de kernelinternals ontbreekt er veel. Iedere keer dat ik

DTrace gebruik, ontdek ik iets anders: wat doet dit ding nu in Godsnaam weer? DTrace leert je het meeste, wanneer je het gebruikt op een machine waarvan jij denkt dat hij niets zou moeten doen. Je ontdekt dat een machine die niets doet, bezig is allerlei dingen te doen waar je niets vanaf weet. We leren dus de hele tijd over Solaris. In bepaalde opzichten zijn het juist de mensen die het langst met Solaris bezig zijn, die het meest leren. We hebben ontdekt dat sommige aannamen met betrekking tot het systeem eenvoudigweg niet kloppen. Een voorbeeld: in Unix kun je een proces forken. Als je dat doet, schep je een nieuw proces en gebruik je de exec system call om een executable image te laden. Er is een tijd tussen de fork en de exec waarin jij en de parent effectief VM pages delen, en daarna kopieer je het en krijg je je eigen kopie. We hebben ons altijd afgevraagd hoeveel page fouten er optreden tussen een fork en een exec. Ik sprak daarover met een senior ontwikkelaar, een heel goed ontwikkelaar, en hij dacht dat het er niet veel waren, vijf à tien. Hij werkte aan een truc die het systeem sneller zou maken, wanneer je maar vijf à tien van die dingen zou nemen. Waren het er echter duidelijk meer, twintig of veertig, dan zou dat het systeem juist langzamer maken. Zijn schema was behoorlijk uitgebreid, en in de tijd dat hij zijn schema aan mij uitlegde, kon ik een script schrijven dat op een productiesysteem dit zou meten. Wij liepen naar een productiesysteem, lieten het een paar minuten draaien en kregen een enorme hoeveelheid resultaten, waaruit bleek dat het er eerder veertig waren. De senior ontwikkelaar zag de data en zei: "Dan is dat het eind van dat idee." Wij hebben dus door DTrace veel van Solaris geleerd en hebben ontdekt dat onze intuïtie over het systeem verbazingwekkend vaak niet blijkt te kloppen. Zoals één van mijn collega's zegt: het meest geweldige aan DTrace is, dat het een eind maakt aan het raden. We hoeven niet meer te raden hoe het systeem werkt, we kunnen het gebruiken om te ontdekken hoe het werkt.'

> 'De tools om software te ontwikkelen zijn in twintig jaar niet wezenlijk veranderd'

BABYSTAPJE *Hoe werkt DTrace, met agents?*

Cantrill: 'Je kunt wel zeggen dat er rocket science is toegepast: agents ja, en nog veel meer. Er is ook gebruik gemaakt van verschillende technologieën. Eén daarvan is, dat we de mogelijkheid hebben om een geoptimaliseerd systeem nemen, en dynamisch op een veilige manier gaandeweg kunnen bespelen. Je hebt dus geoptimaliseerde code en DTrace gaat daar naartoe en verandert de instructies die uitgevoerd worden op een manier die helemaal veilig is. Dat is een beetje link, vooral in de kernel waar de context nogal gevoelig is. We hebben echter het voordeel dat we al tien jaar kernel-development doen en ik begrijp de constraints ervan. Ook de technologie die het mogelijk maakt lopende processen te bewerken, is erg interessant.'

Waarom werkt DTrace met Java-programma's nog niet optimaal?

Cantrill: 'Op dit moment werkt DTrace heel goed met Fortran, C, C++ en zo, min of meer traditionele talen. Voor Java hebben nog maar een klein stapje genomen, omdat we Java nog niet dynamisch kunnen instrumenteren. We kunnen wel een Java-stack backtrace als een actie nemen, je kunt dus activiteiten in het systeem correleren met de Java-stack en zo begrijpen waar je bent in je Java-applicatie. Je kunt volgen waar het programma I/O doet, waar de processor intensief gebruikt wordt, en je kunt DTrace gebruiken om te begrijpen, wanneer mijn Java-applicatie gedescheduled wordt, waar ben ik dan in mijn Java-applicatie? Dat kun je dus op een productiesysteem doen, maar we zien het als een babystapje. Of je het gelooft of niet, het krijgen van een Java-strack backtrace vereiste een grote hoeveelheid technologie. Het is namelijk met een dynamische taal heel moeilijk om uit te zoeken waar je bent binnen die virtuele omgeving, op een manier waar een ontwikkelaar iets aan heeft. De toekomst van DTrace is de mogelijkheid alle software dynamisch te instrumenteren, dus ook dynamische talen als Java, Perl, Python et cetera. Het is een heel erg moeilijke opgave, maar het is waar we ons de komende jaren, voor wat betreft DTrace mee bezig zullen houden. Als je kijkt naar Java, zou je op grond van de *virtual machine* mogen verwachten dat de debuggingtools fenomenaal zijn. Je hebt zoveel vrijheid in een *virtual machine*, die ik in de kernel helemaal niet heb. Gemeten aan die vrijheid, zijn de debuggingtools voor Java relatief primitief.'

Allemaal?

Cantrill: 'Wel er zijn wel een paar interessante dingen, maar niet in de productieomgeving. Laat ik het zo zeggen: we hebben het gevoel dat er ruimte voor ons is waar we zo in kunnen springen om het probleem op te lossen. Het probleem dat we opgelost hebben voor C enzovoorts, is niet adequaat opgelost op Java-gebied. Het is zeer fascinerend, maar ook erg complex. Wat het moeilijk maakt, is dat je onder meer te maken hebt met just-in-time-compilatie, en omdat we het op een volledig veilige manier willen instrumenteren, omdat het in productieomgevingen gebeurt. Ik kan niet vaak genoeg zeggen, hoe belangrijk dat voor ons is: de mogelijkheid

een tool in een productieomgeving te gebruiken. Als je dat kunt, kun je het altijd in een ontwikkelomgeving gebruiken, omdat daar minder constraints zijn. Dat is de toekomst voor DTrace.'

PROTOTYPES *Op welke manier werk je zelf, agile?*

Cantrill: 'We laten mensen de juiste methodologie voor zichzelf vinden. Sun is een erg technicusgedreven bedrijf, wij geven onszelf de vrijheid om heel erg goede mensen te vinden en die geven we de vrijheid. We hebben teams die ontwikkelen gebruikmakend van agile methodes en we hebben teams die andere methodes gebruiken. Bij DTrace geloven we heel sterk in de kracht van zeer kleine, heel gemotiveerde teams. DTrace is in zijn geheel geschreven door drie mensen en wij geloven - en dat is zeker niet gebruikelijk - dat iedereen alles doet. Ik druk het meestal zo uit: je bent verantwoordelijk voor je software van de douche tot het telefoontje. De douche is waar het idee ontstaat en dat gaat via architectuur, implementatie, integratie, prototypes maken, debugging, support en onderhouden. Dat telefoontje is de klant in het veld die belt en zegt: hé, het werkt niet. We zitten niet letterlijk achter de telefoon, maar als er een probleem is met software die ik geschreven heb, zie ik het wel als mijn plicht dat op te lossen. Wij werken dus met heel kleine teams. We zijn ook verantwoordelijk voor de documentatie en het schrijven van de tests. Andere teams werken anders; er zijn teams die veel groter zijn dan DTrace waar het niet noodzakelijkerwijs werkt.'

Zijn er veel requirements veranderd, gedurende het proces?

Cantrill: 'Ja, dat is merkwaardig, als je een besturingssysteem ontwikkelt, dan zit dat bijna altijd in je bloed. Zolang ik me met software bezig houd, heb ik op het laagste niveau van software willen zitten, voor een deel omdat ik graag dichtbij de hardware ben, maar niet zo dichtbij dat je de speelruimte van software verliest. Ik hou van software; er is niets vergelijkbaars in de geschiedenis van het menselijke streven. Als cultuur begrijpen we het nog steeds niet. We hebben geprobeerd het bouwen van software te behandelen op de manier waarop we andere zaken bouwen. Maar het bouwen van een software project is niet zoals het bouwen van een stuwdam of een brug. Het is heel anders. Wat je met software doet, heeft een veel grotere reproduceerbaarheid dan andere gebieden hebben. Wanneer software eenmaal helemaal goed is, dan kan ik die software - het reproduceren ervan kost niets - ergens anders toepassen. Dat is verbazingwekkend, ik hoef niet duizend werklieden in te zetten om ergens die brug opnieuw te bouwen, ik kan mijn bestaande software nemen en die ergens anders gebruiken. Software ligt veel dichter bij wiskunde dan traditionele engineering vakken. Wij begrijpen nog steeds niet helemaal tot op welke hoogte dat waar is. We worstelen daar nog steeds mee. Maar om op je vraag terug te komen: we houden van software en van het vermogen ervan om abstracties te presenteren. Een besturingssysteem laat een realiteit zien en het geeft je een groot gevoel van macht om daar deel van uit te maken. Wanneer je een besturingssysteem ontwikkelt, vooral Solaris, werk je voor een deel op basis van requirements, maar veel van de vernieuwing die je ziet in Solaris 10, is geen reactie op specifieke requirements. Het is niet zo geweest, dat er gezegd is: wij hebben een manier nodig om het systeem dynamisch te instrumenteren. Niet in het minst. Wij zagen de problemen die mensen hadden, en vroegen ons af: is er niet iets dat wij in het besturingssysteem kunnen doen, in deze definitie van realiteit die wij mensen aanbieden, om bepaalde problemen waar ze mee kampen op te lossen. Als gevolg daarvan zijn klanten enthousiast, want zij wisten nooit dat er zoiets uit zou komen. Ze zeggen: ik verwachtte niet dat een besturingssysteem me een development-tool zou opleveren, maar we zijn er heel erg blij mee. Wanneer je een besturingssysteem ontwikkelt, reflecteer je veel over de vraag hoe je mogelijkerwijs het systeem zou kunnen uitbreiden op manieren die de problemen van je klanten helpen op te lossen.'

BOTTLENECKS *Wat is er gedaan om de performance van Java-applicaties te vergroten?*

Cantrill: 'De JVM draait simpelweg als een proces in Solaris. Veel van wat we gedaan hebben, is een antwoord geweest op requirements van het JVM-team. Maar hoewel applicaties nu duidelijk sneller draaien dan bijvoorbeeld op Linux, geloof ik niet dat daar de grootste performance-winst zit. De grootste winst kun je bereiken door in productiesystemen te kijken waar de bottlenecks zitten. Daarom zullen we ons de komende jaren ook op de verder ontwikkeling van DTrace voor Java storten, daar is de meeste winst te halen.'

Dit interview is aanzienlijk langer dan we hier konden weergeven. Een mp3 van de rest van het interview is te vinden op: http://www.array.nl/release

Bryan's DTrace blog:
http://blogs.sun.com/roller/page/bmc

Tekst en fotografie: Dré de Man