Verteilte Compiler-Umgebungen für C/C++ im Vergleich

Ausgeteilt

Große Softwareprojekte kompilieren ist zeitaufwändig und damit vor allem für Firmen teuer. Um den Vorgang zu beschleunigen, nutzen spezielle Compiler-Umgebungen andere Computer im Netzwerk als Rechenknechte. Dieser Artikel testet drei von ihnen und erklärt die dahinter stehende Technik. Daniel Molkentin

Programmierer und Anwender müssen Quellcode nach jeder Änderung neu kompilieren. Bei großen Projekten dauert das oft sehr lange. Den Computer mit mehr RAM oder einer schnelleren CPU ausstatten schafft manchmal Abhilfe. Ist aber der Compiler der Engpass, hilft auch kein Hardware-Upgrade. Der einfachste Weg, den Kompiliervorgang stark zu beschleunigen, sind verteilte Compiler. Sie nutzen die brachliegenden Ressourcen aller Hosts im Netzwerk und verteilen die Rechenaufgaben.

Die drei bekanntesten verteilten Compiler sind Teambuilder [1] von Trolltech, Suses Icecream [2] sowie Distcc [3] von den Samba-Entwicklern. Sie sind keine echten Compiler, sondern dienen als Frontend und nutzen im Hintergrund den GCC. Auf jedem Rechner läuft ein Daemon, der Jobs annimmt.

Die Möglichkeiten für den Einsatz sind vielfältig, doch oft kristallisieren sich typische Szenarien heraus. In den häufigsten wollen die Anwender anderweitig genutzte Computer mit geringer Auslastung einsetzen. Hierbei ist es wichtig, zu gewährleisten, dass zusätzliche Compiler-Jobs die Systeme nicht zu sehr auslasten. Das ist über den Nice-Wert eines Programms möglich.

Gute Implementierungen von verteilten Compiler-Umgebungen überwachen die Systemlast auf den einzelnen Nodes und vergeben an sie je nach Auslastung keine oder nur sehr wenige Jobs. Dieses Modell ist vor allem in Ad-hoc-Farmen nützlich, etwa bei Entwicklerkonferenzen. Beispielsweise war Icecream auf der KDE-Konferenz Akademy durchgehend im Einsatz.

Vielfältige Möglichkeiten

Oft gibt es eine Reihe von dedizierten Maschinen, die speziell zum Kompilieren abgestellt sind und entweder einzelne Jobs per Cron oder bei Bedarf abarbeiten. Das Modell der geschlossenen Build-Farm empfiehlt sich auch aus Sicherheitsgründen. Da sich in den Farmen oft Maschinen mit mehreren Prozessoren befinden, sind die drei Testumgebungen auch dafür gerüstet.

Ein weiteres Einsatzgebiet ist die Embedded-Entwicklung. Da Embedded-Geräte meist mit langsamen CPUs ausgestattet sind, entwickeln die Programmierer auf ihren leistungsstarken Desktops. Nach dem Kompilieren mit Cross-Compilern müssen sie dann das fertige Binary auf das Gerät laden und testen. Mit verteilten Compilern ist es möglich (LAN-Adapter vorausgesetzt), direkt auf den Geräten zu entwickeln und übers Netzwerk zu kompilieren.

Die Jobs teilen sich dann die CPU des Embedded-Systems und des Desktops nach Leistungsvermögen auf. Nur beim Linken ist das Gerät wieder auf sich gestellt, da derzeit keines der drei vorgestellten Programme diese Arbeit verteilt. Nicht zuletzt werden sich die Sofa-Hacker darüber freuen, mit verteilten Compilern ihre Workstation als zusätzlichen Rechenknecht zu verpflichten.

Überwacht

Distcc, Teambuilder und Icecream stellen einen Monitor zur Überwachung des Kompiliervorgangs bereit. Er zeigt die Auslastung aller im Netz vorhandenen Rechner an sowie diverse Statistiken. Außerdem lassen sich sogar Parameter der Netzwerk-Hosts einstellen. Für Systeme ohne Scheduler (wie Distcc, siehe Kasten „Pro und Kontra: Scheduler“) gilt eine Einschränkung: Der Monitor zeigt nur Jobs an, die von dem Host ausgehen, auf dem der Monitor läuft.

Tabelle 1: Alle Kandidaten auf einen Blick

Name	Hersteller	Website	Lizenz	Preis
Teambuilder 1.2	Trolltech	[http://www.trolltech.com]	proprietär	kostenlos für privaten Gebrauch [1]
Icecream (CVS)	Novell/Suse	[http://wiki.kde.org/icecream]	GPL	-
Distcc 2.17	Martin Pool	[http://distcc.samba.org]	GPL	-

[1] Preise für Version 1.2 lagen bei Redaktionsschluss noch nicht in endgültiger Fassung vor.

Da Systeme mit Scheduler all ihre Daten unverschlüsselt übers Netz schicken, raten die Hersteller dazu, den Daemon mit unprivilegierten Accounts in vertrauenswürdigen Umgebungen laufen zu lassen. Außerdem sollte kein Admin die Compiler-Dienste in Richtung Internet öffnen. Abgesehen von Sicherheitsrisiken ist diese Idee auch mit Blick auf die Geschwindigkeit selten sinnvoll. Selbst mit Kompression sollten die beteiligten Rechner mit mindestens 10 MBit/s vernetzt sein. Andernfalls verschwindet der Zeitgewinn, der durch das Verteilen erreicht wird, ganz schnell.

Alle zugleich

Die meisten Softwareprojekte benutzen das Programm »make« zum Kompilieren. Damit Make mehrere Jobs parallel ausführt, verwendet der Benutzer den Schalter »-j« und beschränkt ihn durch ein sinnvolles Maximum. Sind in der Compile-Farm etwa zehn PCs mit je zwei Prozessoren vorhanden, ist ein Wert von 10 bis 15 sinnvoll. Nur bei Distcc bringen Werte bis zu 25 noch eine geringe Steigerung. Generell skaliert dieser Wert jedoch nicht unendlich nach oben, die Testergebnisse (siehe Abbildungen 8 und 9) belegen das.

Das Hauptproblem bei der Skalierung sind Abhängigkeiten von Automake: Größere Projekte strukturieren ihren Code in Unterverzeichnissen und definieren in den zugehörigen Makefiles für jedes Verzeichnis so genannte Targets. Make kompiliert zunächst alle Dateien (»*.c«, »*.cpp«) in einem Verzeichnis und linkt sie dann gegen die nötigen Bibliotheken. Erst danach wechselt der Prozess in die Unterverzeichnisse. Liegen in jedem Target nur zehn Dateien, verarbeitet Make auch nur diese zehn gleichzeitig, da das Linken lokal stattfindet und die Objektdateien vorher kompiliert sein müssen.

Das KDE-Projekt begegnet diesem Problem mit dem Programm Unsermake. Es erlaubt eine weiter gehende Parallelisierung des Build-Prozesses, indem es alle Targets relativ zum obersten Verzeichnis definiert. Damit vermeidet Unsermake rekursive Aufrufe. Es erreicht eine recht gute Parallelisierung, da es die nicht verteilbaren Aufgaben wie das Linken der Objektdateien so weit wie möglich hinauszögert.

Teambuilder

„Wir entschuldigen uns bei den Entwicklern dafür, dass sie jetzt weniger Zeit haben im Netz zu surfen, Quake zu spielen oder ihre Mitarbeiter mit Scherzpistolen zu beschießen“, kündigte die Firma Trolltech ihr neues Produkt Teambuilder im April 2001 an. Mit dessen Hilfe soll sich das Kompilieren bis zu 14fach beschleuningen lassen – ganz ohne großen Aufwand. Ein zentraler Scheduler verteilt die Jobs.

Das Produkt entstand aus dem eigenen Leidensdruck heraus, oft große Mengen C++-Code übersetzen zu müssen. Nachdem ein erster Prototyp fertig war, entschloss sich Trolltech dazu, Teambuilder als Produkt zu vermarkten [1]. Die Anspielung auf die einfache Administration war wohl ein Seitenhieb auf Open Mosix, das zwar Prozesse verteilt, aber aufwändig zu installieren ist und einen modifizierten Kernel erfordert. Daher ist es nicht für den Ad-hoc-Betrieb etwa in einer Konferenz geeignet.

Teambuilder ist proprietäre Software, deren Kosten für Firmen zwar gering sind, Privatpersonen und freie Projekte können sich Teambuilder jedoch meist nicht leisten. Für den privaten Gebrauch und mit einer Beschränkung auf drei Knoten ist daher die Personal Edition kostenlos von der Website zu haben [6]. Die Preise für die kommerzielle Version bemessen sich an der Anzahl der Hosts. Dabei zählen sowohl Job-verarbeitende als auch Job-generierende Hosts.

Als Grundlage für den Test dient eine zeitbeschränkte Ausgabe der Version 1.2. Das Tar.gz-Archiv enthält eine kurze

So funktioniert's

Alle hier getesteten Kandidaten arbeiten (abgesehen vom Scheduler) nach demselben Prinzip. Will ein Host seine Kompilierarbeiten verteilen, übernimmt er zuerst den Präprozessorlauf (zum Beispiel mit »gcc -E«). Dabei kopiert der Compiler alle Header in die Quelldateien und sendet die Quellen an einen oder mehrere Zielrechner. Diese kompilieren alle erhaltenen Quellen und liefern Objektdateien zurück, die der Host sammelt und gegen die nötigen Bibliotheken linkt.

Der Trick mit den Symlinks

Jeder Benutzer installiert auf seinem Rechner ein Client-Programm, das per symbolischem Link die Namen »gcc«, »g++« und so weiter bekommt. Es muss im Pfad vor den tatsächlichen Compilern liegen. Ruft »make« den »gcc« auf, startet tatsächlich das Client-Programm, das sich dann um die Verteilung der Jobs kümmert.

Diese Technik ist nicht neu und wird auch schon von anderen GCC-Erweiterungen wie Colorgcc genutzt, das alle Ausgaben des GCC einfärbt. Außer mit Colorgcc lassen sich Distcc, Teambuilder und Icecream mit dem Compiler Cache (Ccache) [1] von Andrew Tridgell kombinieren. Er speichert alle Objektdateien temporär, die der Compiler dann nach einem »make clean« nicht neu übersetzen muss. Falls sich etwas am Source geändert hat, kümmert sich Ccache darum, keine alten Dateien auszuliefern.

Kaskaden

Um Ccache zusätzlich mit einem der vorgestellten Programme zu kombinieren, genügt es, wie oben beschrieben symbolische Links mit den Namen der Compiler in einem dedizierten Verzeichnis anzulegen, das der Benutzer im Pfad ganz nach vorne stellt, zum Beispiel so: »PATH=/opt/ccache/bin:/opt/distcc/bin:/usr/bin«

Ein Aufruf von »g++« startet Ccache, das seinerseits mit »g++« das Client-Programm des verteilten Compilers startet. Dieses ruft schließlich den echten »g++« auf. Bei Teambuilder ist übrigens das Zwischenschalten von Colorgcc nicht notwendig, falls die Umgebungsvariable »TEAMBUILDER_COLORIZE=1« gesetzt ist. Dann erledigt der Daemon das Einfärben ganz von selbst.

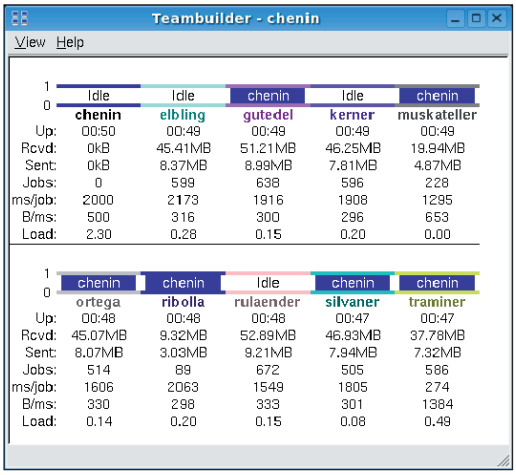

Abbildung 1: Der Teambuilder-Monitor bietet allerlei Statistiken über die Hosts im Netzwerk und deren Jobs.

Anleitung und eine Installationsdatei. Teil des Pakets ist außerdem ein statisch gelinkter Monitor, der nicht nur die Auslastung aller Maschinen überwacht (siehe Abbildung 1), sondern auch Parameter einzelner Knoten konfiguriert, wie in Abbildung 2 zu sehen ist.

Einfache Installation

Version 1.2 verteilt gegenüber der ersten Release nicht nur die Last besser, sondern ist auch performanter. Außerdem haben die Entwickler den Umgang mit heterogenen Umgebungen verbessert; es ist nun möglich, Linux und andere Unix-Plattformen zu mischen.

Die Installation des Programms ist textbasiert und verlangt vom Benutzer lediglich die Lizenzbedingungen zu akzeptieren und den Installationspfad zu bestätigen. Teambuilder wählt standardmäßig »/opt/teambuilder«. Keinesfalls sollten Nutzer hier »/usr« wählen, da die Installationsroutine dann eventuell die echten GCC-Binaries durch symbolische Links ersetzt (Näheres erklärt der Kasten „So funktioniert's“).

Die Dokumentation ist – wie von Trolltech gewohnt – umfangreich und liegt im HTML-Format vor. Sie erklärt die grundsätzlichen Schritte, die Verwendung des Monitors und erweiterte Einstellungen sowie Tuningmöglichkeiten. Da die Einbindung von Teambuilder manuell erfolgt, schildert die Anleitung sie sehr detailliert.

Ein Startskript für Daemon und Scheduler liegt bei und benötigt lediglich ein paar Anpassungen an die distributionsspezifischen Eigenheiten. Über die beiden Dateien »/etc/tbscheduler.conf« und »/etc/tbdaemon.conf« sind die Daemons zu konfigurieren. Alternativ ist die Konfiguration über Umgebungsvariablen möglich.

Die Teambuilder-Komponenten kommunizieren über die beiden Ports 10247 (TCP) und 10248 (UDP), die auf allen eingesetzten Maschinen geöffnet sein müssen. Bei eventuell vorhandenen Firewalls sind diese Ports vorher freizuschalten.

Nach Herstellerangaben funktioniert Teambuilder sowohl mit den mittlerweile in die Jahre gekommenen GCC-Versionen 2.95 und 2.96 als auch mit 3.0 und 3.2. Nach den Erfahrung des Autors bereitet auch GCC 3.3 keine Probleme. Obwohl Teambuilder laut Dokumentation GCC unter FreeBSD und Solaris 7 sowie den Irix Mips Pro Compiler in der Version 7.7.1.1m unterstützt, ist das Programm bisher nur für Linux verfügbar. Interessenten mit anderen Plattformen wenden sich an Trolltech.

Diese sorgfältige Abgrenzung scheint übertrieben angesichts der Tatsache, dass jeder C/C++-Compiler mit seiner Umgebung auf nahezu identische Art und Weise kommuniziert. Laut Auskunft des zuständigen Produktmanagers hat Trolltech den Teambuilder mit fast allen kommerziellen Unix-C-Compilern erfolgreich getestet. Auch die meisten C++-Compiler sollten funktionieren.

Suns Forte-Compiler jedoch steht einer Verteilung von Jobs im Wege, da er Templates auf besondere Weise behandelt. Das trifft allerdings nicht nur auf Teambuilder zu, sondern auf alle drei getesteten Systeme.

Cross Compiling und die Unterstützung von inkompatiblen Compiler-Versionen beherrscht Teambuilder zwar, jedoch erst nachdem der Benutzer das Betriebssystem und die Compiler-Version in die Datei »/etc/tbdaemon.conf« eingetragen hat. Diese gibt er anschließend per Umgebungsvariable vor dem Kompiliervorgang an.

Distcc

Etwa zur gleichen Zeit wie Trolltech kam der Australier Martin Pool auf die Idee, GCC-Jobs über das Netzwerk zu verteilen. Obwohl der Ansatz dem von Teambuilder ähnelt, unterscheiden sich die Implementierungen deutlich. Pools Distcc [3] arbeitet zum Beispiel ohne Scheduler und steht unter der GPL. Neben C und C++ testet das Entwicklerteam seinen Compiler auch für Objective C und Objective C++, was vor allem für Mac-OS-X-Entwickler interessant ist. Die meisten aktuellen Distributionen liefern Distcc-Pakete mit.

Es existieren mehrere Frontends für den Distcc, die getestete Version brachte einen GTK-Client mit, den Abbildung 3 darstellt. Außerdem gibt es ein KDE-Frontend namens »distcc-monitor« und einen Konsolen-basierten Monitor. Alle drei Applikationen sind aber nur sehr rundimentär und bieten kaum Zusatzinformationen über die Hosts.

Aus Sicherheitsgründen gibt der Daemon »distccd« nach dem Start seine Root-Privilegien ab und wechselt die User-ID zum Benutzer »distcc«. Wer Distcc aus den Quellen baut, sollte daher unbedingt vorher einen solchen Benutzer anlegen, die Distributionen erledigen das in der Regel beim Installieren des Pakets. Seit Version 2.1 unterstützt Distcc

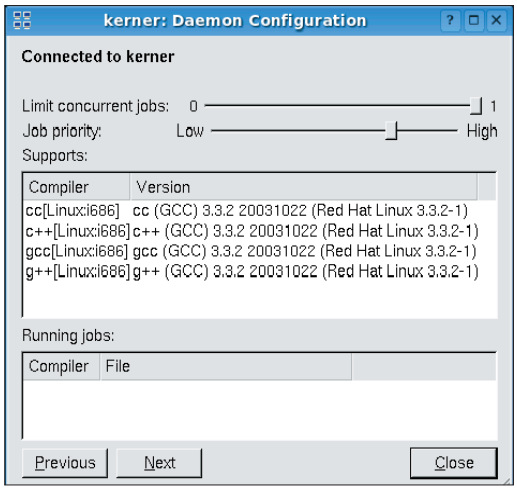

Abbildung 2: Mehr als nur überwachen: Vom Teambuilder-Monitor aus lassen sich Einstellungen an den Daemons vornehmen.

Abbildung 3: Der GTK-basierte Distcc-Monitor des Gnome-Projekts zeigt prinzipbedingt nur jene Jobs an, die von dem eigenen Rechner ausgehen.

SSH-geschützte Verbindungen. Diese Fähigkeit verdankt er seiner dezentralen Administration. Seine beiden Konkurrenten Teambuilder und Icecream sind dazu nicht in der Lage.

Kein Scheduler

Bevor die Programmierer ihre Jobs mit Distcc verteilen, geben sie dem Daemon alle Rechenknechte in der Umgebungsvariablen »DISTCC_HOSTS« an. Der Kasten „Pro und Kontra: Scheduler" erklärt die Details. Laut Anleitung starten Benutzer nun die Kompilierung für C-Programme mit dem Aufruf »make -j*Anzahl* CC=distcc«. Die elegantere Variante ist jedoch, wie im Kasten „So funktioniert's" beschrieben, ein Dummyverzeichnis, das symbolische Links auf »distcc« enthält. Diese Variante funktioniert auch mit Programmen, die C und C++ im Wechsel benötigen:

```
ln -s distcc   cc
ln -s distcc   gcc
ln -s distcc++ c++
ln -s distcc++ g++
```

Icecream

Ende vergangenen Jahres machten sich die Suse-Mitarbeiter Stephan Kulow, Michael Matz und Cornelius Schumacher daran, Distcc in eine andere Richtung weiterzuentwickeln. Sie waren mit der proprietären Lizenz von Teambuilder nicht zufrieden und Distcc taugte nicht für die speziellen Bedürfnisse von Suse mit ihren vielen unterschiedlichen Compilern und Hardware-Architekturen. Den GPL-lizenzierten Fork, der den Namen Icecream [2] trägt, portierten sie nach C++. Mit einem Scheduler ausgestattet ähnelt er in seiner Funktionsweise nun eher Teambuilder. Den für Distcc entwickelten Distcc-Monitor passten die Programmierer ebenfalls an und nannten ihn Icemon.

Schicker Monitor

Das KDE-Team ist stark an der Programmierung von Icecream beteiligt und hat ihn in das Entwicklungs-Verzeichnis »kdenonbeta« aufgenommen. Wer Icecream kompiliert, sollte – wie schon beim Teambuilder – als Präfix nicht »/usr« wählen, am sichersten ist »/opt/icecream«. In »/opt/icecream/bin« befinden sich nach der Installation der Daemon »iceccd«, der Client »icecc« mit den entsprechenden symbolischen Links sowie der Scheduler. Init-Skripte sowie Spec-Dateien für Suse liegen den Sourcen bereits bei.

Der Icemon-Monitor ist das Schmuckstück von Icecream. Er unterstützt mehrere Darstellungsmodi, von denen die Sternansicht die eindrucksvollste ist, wie Abbildung 6 zeigt. Die leider relativ CPU-hungrige Gantt-Ansicht zeigt auch die Dauer eines einzelnen Jobs an. In der Übersichtsliste (Abbildung 7) listet Icemon alle Hosts untereinander auf. Die diversen Ansichten des Monitors leiden aber noch an Kinderkrankheiten. So eignet sich die Übersicht nicht für Far-

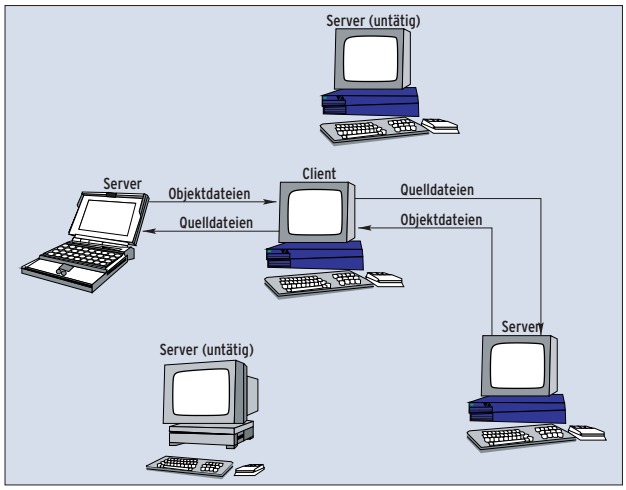

Abbildung 4: Der Distcc arbeitet ohne Scheduler. Vor allem in heterogenen Netzwerken steigt daher der Administrationsaufwand.

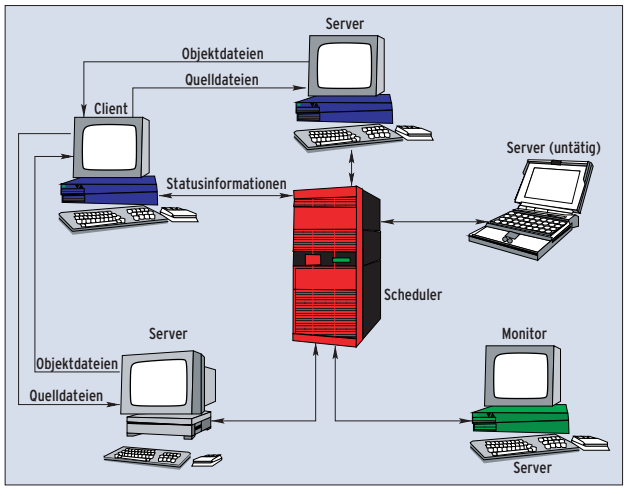

Abbildung 5: Teambuilder und Icecream setzen auf einen Scheduler, der die komplette Administration der Compile-Farm übernimmt.

men mit vielen Hosts, da die Liste sehr lang und unübersichtlich wird.
Icecream ist optimal darauf vorbereitet, verschiedene Compiler-Ausgaben zu verarbeiten. Es ist nicht nötig, auf allen Hosts kompatible Versionen zu installieren. Das beiliegende Skript »create-env« packt alle nötigen Dateien in ein »tar .bz2«-Archiv mit zufälligem Namen. Es empfiehlt sich, es sinnvoll umzubennen, etwa in »i386-3.3.1.tar.bz2«.
Benutzer teilen Icecream den Dateinamen in der »ICECC_VERSION«-Umgebungsvariablen mit. Ist sie vorhanden, übergibt Icecream den kompilierenden Daemons neben den Quelldateien auch das Archiv. Dies packen die Daemons in einer Chroot-Umgebung aus und arbeiten darin. Dazu benötigt der laufende »iceccd« aber Root-Rechte. Das Konzept lässt sich sogar zum Cross-Kompilieren einsetzen. Die Syntax für »ICECC_VERSION« lautet dann »*Nativer-GCC*.tar.bz2, *Plattform:Cross-GCC*.tar.bz2«.

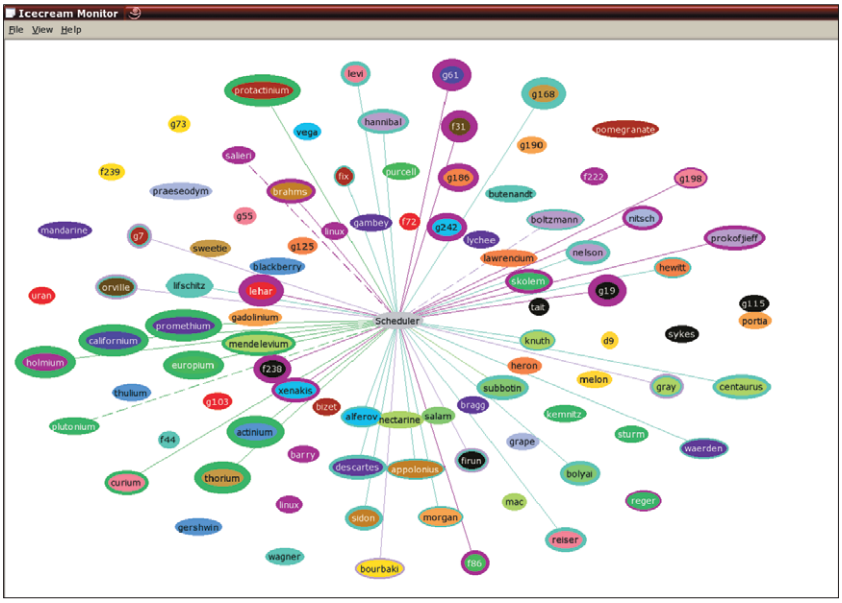

Abbildung 6: In der Sternansicht von Icemon sind alle beteiligten Rechner zu sehen. Die Ellipsen um die Hosts herum zeigen deren Auslastung an. In diesem Modus hat der Anwender eine optimale Übersicht.

Test-Parcours

Für die Geschwindigkeitstests mussten sich alle Probanden in zwei Kategorien bewähren: Im ersten Gang ging es ans Kompilieren eines Vanilla-Kernels, Version 2.6.7. Danach war das Kompilieren von C++-Code an der Reihe; die Compiler bekamen mit den KDE-Kernbibliotheken (KDE-Libs) eine Menge zu tun. Die Wahl fiel auf die KDE-Libs, da Unsermake nur die KDE-Build-Umgebung unterstützt. So konnten die Systeme zeigen, wie sie die von Unsermake geschaffene Parallelisierung nutzen.
Als Testhardware kamen zehn Athlon XP 2800 mit einem Takt von 2080 MHz und 1024 MByte Arbeitsspeicher zum Einsatz. 16 MByte des RAM reservierte sich die Grafikkarte. Alle Rechner waren über ein geswitchtes 100 MBit-Netzwerk verbunden. Sowohl an der Hard- als auch an der Software gab es keine Tuning-Eingriffe. Mit Kniffen wie beispielsweise dem Setzen von Nice-Werten wäre vermutlich noch mehr Geschwindigkeit herauszuholen, doch ist es die Entscheidung des Admin, ob seine Maschinen noch einen anderen Zweck erfüllen sollen und wie viel Leistungsvermögen er vorübergehend entbehren kann. Vor allem menschliche Benutzer teilen ihren PC nur ungern mit allzu gefräßigen Hintergrundprozessen. Gegebenenfalls ist zu empfehlen, den Ressourcenanspruch des Compilers einzuschränken.

Pro und Kontra: Scheduler

Teambuilder und Icecream benutzen einen Scheduler. Dieser Daemon-Prozess läuft auf einem beliebigen Host und überwacht die Auslastung aller verfügbaren Rechner. Ein Daemon, der Aufgaben verteilen will, fragt beim Scheduler an, welche Rechner er zur Auswahl hat. Anhand der Auslastung weist der Scheduler ihm dann Hosts zu. Abbildung 4 zeigt, wie ein System ohne Scheduler arbeitet, Abbildung 5 illustriert den Scheduler-Ansatz.
Dass der Distcc keinen Scheduler verwendet, hat mehr Nach- als Vorteile. Da sich verfügbare Knoten nirgendwo registrieren, teilen Benutzer dem Daemon die Liste aller Knoten über die Umgebungsvariable »DISTCC_HOSTS« mit. Diese muss auf jedem Host gesetzt sein. Sie enthält – durch Leerzeichen getrennt – die Namen oder die IP-Addressen aller Rechner in der Compile-Farm. Die Distcc-Autoren empfehlen, die stärksten Rechner zuerst einzutragen. Neben den administrativen Aufgaben ist ein Scheduler gewöhnlich für die Verwaltung unterschiedlicher Betriebssysteme und Compiler auf Quell- und Zielplattformen zuständig. Er verhindert so, dass aus Versehen eine PowerPC-Maschine Jobs für einen Intel-GCC zugewiesen bekommt.

Inkompatible ABIs

Dieses Feature ist selbst in einer reinen Intel-Build-Farm wichtig. Denn das ABI (Application Binary Interface, die interne Struktur der kompilierten Objektdateien also) hat sich bei C im Laufe der letzten Jahre zwar praktisch nicht geändert und es lassen sich sogar Objektdateien verschiedener Compiler miteinander linken. Doch bei C++ kam es allein beim GCC seit Version 2.95 mehrfach zu inkompatiblen ABI-Änderungen.
Ist ein Scheduler vorhanden, dann genügt es, bei Quell- und Zielplattform jeweils Compiler und Betriebssystem anzugeben (meist erkennt das Compile-System diese auch automatisch). Distcc unterscheidet allerdings nicht zwischen verschiedenen Versionen und setzt voraus, dass alle beteiligten Maschinen eine identische Kombination aus Compiler und Betriebssystem fahren.

Scheduler für Distcc

Auch wenn die Autoren von Distcc beteuern, ein Scheduler brächte kaum mehr Effizienz und sei sogar ein Single Point of Failure, reduziert eine zentrale Struktur den Administrationsaufwand vor allem in heterogenen Umgebungen enorm.
Wer trotzdem unbedingt Distcc mit einem Scheduler verwenden möchte, sei auf Distcc-Proxy [4] verwiesen, gleichzeitig jedoch gewarnt: Der Einsatz der Software bedeutet doppelt so hohen Netzverkehr, da alle Daten immer erst den Umweg über den Proxy nehmen. Dieser verteilt sie dann an die Hosts.

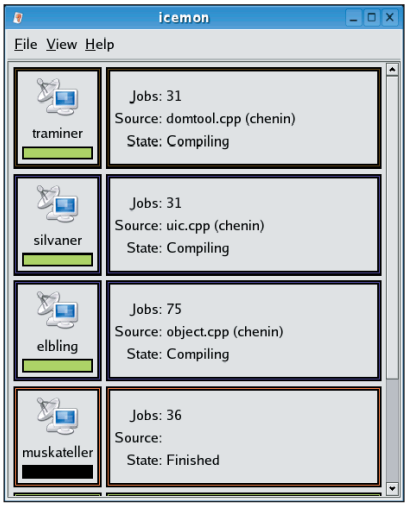

Abbildung 7: Diese Ansicht des Icemon wird bei vielen Rechnern im Netz sehr unübersichtlich.

Bei der Analyse der Ergebnisse in den Abbildungen 8 und 9 zeigt sich der deutliche Geschwindigkeitsgewinn bei verteilten Compilern. Die Kernelübersetzung dauert statt 17 nur noch 2,5 Minuten. Icecream ist deutlich am schnellsten, fast siebenmal schneller als ein einzelner GCC. Teambuilder ist der Langsamste im Rennen.

Die KDE-Libs kompilieren nur dreimal schneller. Auch hier liegt Icecream vorne, jedoch nicht so deutlich. Distcc ist der langsamste Kandidat. Unsermake bringt noch einmal einen Geschwindigkeitsschub. Bei der genauen Beobachtung des Build-Vorgangs ist zu sehen, dass die Kompilierung nach einem Drittel der Gesamtzeit abgeschlossen ist, der Rest geht fürs Linken drauf.

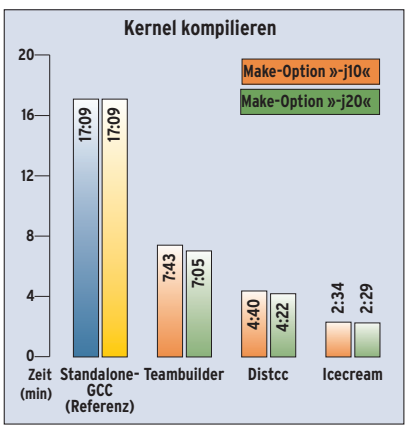

Abbildung 8: Die verteilten Compiler nutzten jeweils zehn Systeme parallel. Der GCC lief mit der Make-Option »-j1« beziehungsweise »-j4«. Icecream ist fast siebenmal schneller als ein Standalone-System. Die Option »-j20« bringt kaum mehr Performance.

Die Objektdateien linken ist vor allem bei den KDE-Libs kaum effizient parallelisierbar, da sehr viele Bibiliotheken übers Netz gesendet werden müssten. Zudem enthalten aus C++-Code generierte Objekt-Dateien viel mehr Symbole als C-Dateien. Mit dieser Flut kann der auf C spezialisierte GNU Linker nur sehr schlecht umgehen. Für verteiltes Kompilieren sollten Anwender bisweilen also mit dem Linker sparsam umgehen, so weit es möglich ist.

Übermäßiger Gebrauch des »-j«-Parameters bringt meist kaum Vorteile. Ebenso liefert der Einsatz von »-j« auf einem Einprozessorsystem entgegen weitverbreitetem Glauben keine Verbesserungen. Bei SMP- und Hyperthreading-Maschinen führt es jedoch zu erheblich besseren Zeiten.

Keine Wunder

Eine 14fache Geschwindigkeit – wie von Trolltech – versprochen lieferte keines der Systeme. Mit einem hochparallelisierten Build-System und mehr als den hier getesteten zehn Knoten sind allerdings noch Steigerungen möglich. Eine schwache CPU auf dem Host und starke Rechenboliden im Netzwerk könnten zu Werten um 14 führen.

Ein Einbruch der Geschwindigkeit ist beim Distcc zu erwarten, sobald mehr als ein Anwender die Build-Farm nutzt. Das liegt daran, dass kein Scheduler vorhanden ist. Die Nutzer müssten sich auf eine zufällige Reihenfolge der Hosts in der »DISTCC_HOSTS«-Variablen einigen, um Jobs gerechter zu verteilen. Bei mehreren Anwendern ist daher grundsätzlich eines der beiden Scheduler-basierten Systeme zu empfehlen.

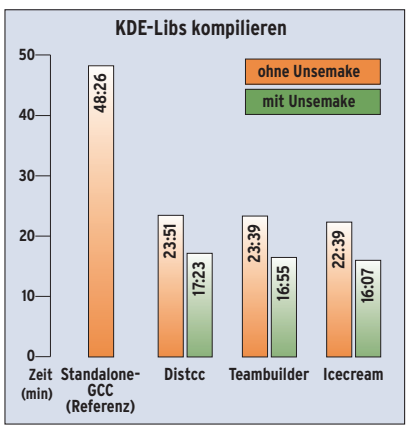

Abbildung 9: Auch hier standen zehn Systeme parallel zur Verfügung. Wieder ist Icecream am schnellsten, gut dreimal schneller als ein einziger GCC alleine. Unsermake sorgt für noch einmal fast 30 Prozent mehr Geschwindigkeit.

Fazit

Das Teambuilder-Paket wirkt von allen getesteten am rundesten, lässt sich einfach anpassen und ist gut dokumentiert. Der enorme Ausreißer beim Kompilieren des Kernels überraschte jedoch unangenehm. Anpassungen der Default-Einstellungen sollten Teambuilder jedoch deutlich weiter vorne platzieren. Nicht zu unterschätzen ist, dass Kunden Anspruch auf Support erhalten. Gerade bei komplizierteren Setups ist das ein absoluter Pluspunkt für Teambuilder. Sein Monitor bietet zwar nur eine Ansicht, der Zustand der Build-Farm lässt sich in ihr aber schnell erfassen

Von einigen Kinderkrankheiten abgesehen ist Icecream eine solide Alternative zu Teambuilder. Doch auch wenn sich das Programm nach Entwickleraussagen schon für den Produktiveinsatz eignet, wirkt vieles nicht so ausgereift wie beim Trolltech-Pendant. Icecream ist noch relativ jung und entwickelt sich in rasantem Tempo. Während des Testzeitraums rundeten die Programmierer das System zunehmend ab. Es ist zu erwarten, dass Icecream in Kürze viele Verbesserungen erhalten wird.

Verteiltes Kompilieren spart letztlich nicht nur Zeit und Geld, es ist auch einfach zu realisieren. Und so lange keines der Tools auf Windows portiert ist, gehen Linux-Entwickler weiter ihrer Lieblingsbeschäftigung nach, während die Windows-Kollegen frustriert auf ihren Compiler warten. *(mwe)* ■

Infos

[1] Teambuilder: [http://www.trolltech.com/products/teambuilder/]
[2] Icecream: [http://wiki.kde.org/icecream/]
[3] Distcc: [http://distcc.samba.org]
[4] Compiler Cache: [http://ccache.samba.org]
[5] Distcc-Proxy: [http://lists.samba.org/archive/distcc/2002q4/000444.html]
[6] Teambuilder Evaluation: [http://www.trolltech.com/download/teambuilder/evaluate.html]