Bessere Buildperformance mit ccache und icecc

C++/C/Objective-C/C++ Projekte verteilt kompilieren, Kaffeepausen verkürzen

Johannes @joke@kif.rocks
13.02.2025

Problem

- Große Projekte bauen dauert und nervt
- Nicht geänderte Codebausteine müssen bei Änderungen mitgebaut werden
- `Geteilte Codebasis wird mehrfach gebaut`

Lösungsansätze

- Speicher ist billig und Code ändert sich selten vollständig
- Müsste es nicht möglich sein Ergebnisse wiederzuverwenden?
- Entwicklerteam hat alles in einen Netz, Buildinfrastruktur besteht aus vielen Knoten (Cloud/Rechenzentrum/CI-Pipeline), kann man das nicht ausnutzen?
- Lösung wrapper um Compiler
- ccache:
    - Datei noch nicht in Cache? Datei mit Metadaten hashen, compilieren, im Projekt und Cache ablegen
    - Datei im Cache? Aus Cache ins Projekt
- icecc:
    - Zentraler Scheduler macht Lastverteilung
    - Wrapper auf Clients schickt Ergebnisse an Clients

Ccache: Installieren und einrichten

- sudo apt-get install ccache # oder wie auch immer
- ccache -M 10G # Cache vom Default (5 GB) auf 10 GB erhöhen
- sudo ln -s /usr/bin/ccache /usr/local/bin/gcc
- sudo ln -s /usr/bin/ccache /usr/local/bin/g++
- Unter Debian und co: sudo ln -s /usr/lib/ccache/* /usr/local/bin
- Das reicht für Builds auf der eigenen Kiste für die eigene Kiste.
- Prefix Alternative für cross-Compiling etc
  - ccache gcc $CFLAGS
  - Muss dann entsprechend in den Makefiles/Cmakedateien what ever notiert werden

Weitere Optionen und Aufräumen

- Per ccache --set-config=option=wert
- Zum Beispiel:
  - cache_dir für anderes Cacheverzeichnis
  - sloppiness um „Hit-Rate“ zu erhöhen
  - compression → Default false, falls Speicher knapp ist ändern

- Aktuelle Config ausgeben:

  ccache –-print-config | ccache -p
- ccache räumt sich selbst auf, händisch mit

  ccache –-cleanup | ccache -c
- Komplettes leeren:

  ccache –-clear | ccache -C

Statistiken

- ccache –-show-stats | ccache -s
- Zurücksetzen mit ccache –-zero-stats | ccache -z

```
cache directory                     /usr/local/ccache/
primary config                      /usr/local/ccache//ccache.conf
secondary config      (readonly)    /etc/ccache.conf
cache hit (direct)                     7
cache hit (preprocessed)               1
cache miss                         14076
cache hit rate                      0.06 %
called for link                       43
called for preprocessing            2654
unsupported code directive             3
no input file                        943
cleanups performed                     0
files in cache                     42171
cache size                          4.8 GB
max cache size                     10.0 GB
```

Praktisches Beispiel

- Linux Kernel 4.9.0.4
- 718 MB Sourcecode
- Übersetzen der Debian .config mit time make -j4

```
real    51m55,577s
user   183m27,728s
sys    10m44,220s
```

- Erneutes Übersetzen:
  - make clean
  - time make -j4:

```
real 6m9,393s
user    15m37,012s
sys  1m33,652s
```

Praktisches Beispiel

- ccache -s

```
cache directory                     /usr/local/ccache/
primary config                      /usr/local/ccache//ccache.conf
secondary config      (readonly)    /etc/ccache.conf
cache hit (direct)                14016
cache hit (preprocessed)             26
cache miss                        14052
cache hit rate                    49.98 %
called for link                      49
called for preprocessing           5281
unsupported code directive            6
no input file                       949
cleanups performed                    0
files in cache                    42154
cache size                         4.8 GB
max cache size                      10.0 GB
```

Weitere Anwendungsmöglichkeiten / Verteilter Ccache

- Jenkinsbauten beschleunigen
- Continous integration
- Teilen mit Kollegen?
  - Theoretisch per NFS Share, aber man ccache sagt:

SHARING A CACHE ON NFS

It is possible to put the cache directory on an NFS filesystem (or similar filesystems), but keep in mind that:

- Having the cache on NFS may slow down compilation. Make sure to do some benchmarking to see if it’s worth it.
- ccache hasn’t been tested very thoroughly on NFS.
- A tip is to set temporary_dir to a directory on the local host to avoid NFS traffic for temporary files.

Besser: Mit anderen Tools teilen

- icecc/distcc arbeiten auch als Wrapper
- Verteilen Last auf Knoten im Netzwerk
- Ermöglichen verteiltes Bauen
- Lässt sich über den Prefixtrick mit ccache zusammen aufrufen
- Oder mit Buildsystemen, die ähnliches machen
- Immer: Testen, ob Kombination nicht mehr schadet als nützt!

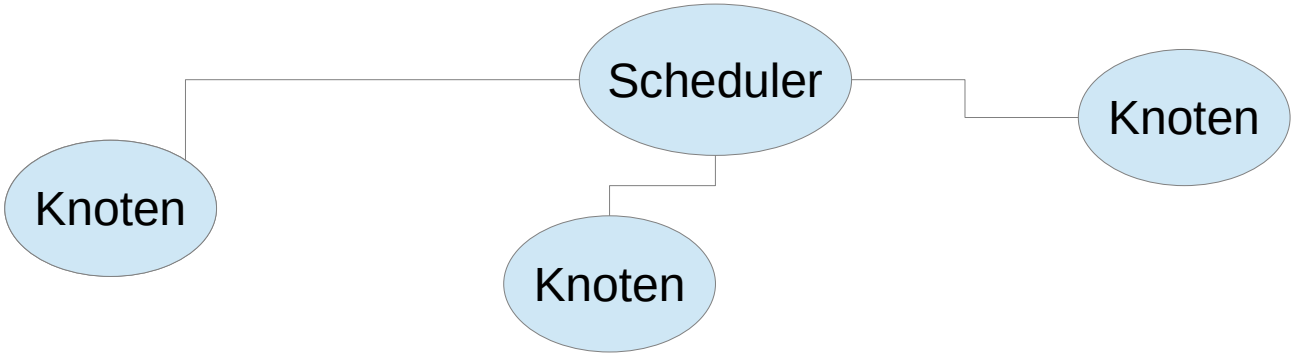
icecc

- Große Projekte bauen dauert und nervt
- Nicht geänderte Codebausteine müssen bei Änderungen mitgebaut werden
- Geteilte Codebasis wird mehrfach gebaut
- Ccache hilft zwar, kostet aber Speicherplatz und zum initalen Befüllen hat man trotzdem nur die Kerne des eigenen Rechners

Lösung: icecc/icecream

- Idee: Baujobs an andere Rechner auslagern
- Wrapper um Compiler
- Fork von distcc
- Unterstützt Linux/MacOS X, FreeBSD, Dragonfly BSD
- Unterstützt clang/gcc für C/ C++ und ObjectiveC
- Arbeitsweise
  - Scheduler verteilt Bauaufträge an Knoten im Netzwerk
  - Kompilieren erfolgt im Netzwerk
  - Preprozessing und Linken passiert lokal
  - Knoten können Client UND Server sein, oder auch nur Client

Netzstruktur



Icecc: Installieren und einrichten (Debian)

- sudo apt-get install icecc # oder wie auch immer
- Theoretisch werden mehrere Scheduler im gleichen Netzwerk unterstützt, praktisch haben ältere Clients damit Probleme.
- Also: Nur ein Scheduler (muss nicht server oder client sein!) und überall die gleiche Version!
- /etc/icecc/icecc.conf anpassen:
  ICECC_NETNAME=stratum-cluster #praktisch bei vpn oder mehreren clustern

  ICECC_SCHEDULER_HOST=icecc-scheduler.example
- sudo ln -s /usr/bin/icecc /usr/local/bin/gcc
- sudo ln -s /usr/bin/icecc /usr/local/bin/g++
- Auf Scheduler host: Scheduler starten, z.B. /etc/init.d/icecc-scheduler start
- Auf Knoten: /etc/init.d/iceccd start

Icecc: Installieren und einrichten

- Aufruf des clients entweder über symlink auf compiler oder als Präfix:
- sudo ln -s /usr/bin/icecc /usr/local/bin/gcc
- sudo ln -s /usr/bin/icecc /usr/local/bin/g++
- Unter Debian: sudo ln -s /usr/lib/icecc/bin/* /usr/local/bin/
- Oder eben als Aufruf via icecc gcc hello.c
- Für native Builds reicht das: Icecc erkennt wenn er noch keine Toolchain hat und erstellt sie automagisch und reicht sie via Scheduler an die Knoten durch, wo sie gecached werden

Crosscompiling/Verschiedene Targets etc

- Für jede Plattform muss eine entsprechende Toolchain als tgz vorliegen
- Native Toolchain: icecc –build-native
  adding file /bin/true #snip
  creating 7035202699cf39e9f581a5a3630f5c88.tar.gz
- mv 7035202699cf39e9f581a5a3630f5c88.tar.gz icecc-native.tar.gz
- Für andere, z.B arm64:
  icecc-create-env --gcc aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc aarch64-linux-gnu/bin/aarch64-linux-gnu-g++
  icecc-create-env --clang aarch64-linux-gnu/bin/aarch64-linux-gnu-gcc aarch64-linux-gnu/bin/aarch64-linux-gnu-g++
  mv bf58716cd90905bd59a6604859db572f.tar.gz icecc-aarch.tar.gz

Crosscompiling/Verschiedene Targets etc

- Mit Umgebungsvariabe ICECC_VERSION werden die zu verwendenden Toolchains konfiguriert:
- ICECC_VERSION=<native_filename>(,<platform>:<cross_compiler_filename>=<target>
- Plattform: Plattform der Buildmaschine, z.B: i386
- Cross_compiler_filename: Pfad zum toolchain.tar.gz
- Target: Präfix des Compiler-Binary z.B. aarch64-linux-gnu für aarch64-linux-gcc und aarch64-linux-g++
- Beispiel aus der Praxis:

  ICECC_VERSION: "/home/joke/icecc-native.tgz,/home/joke/gcc-linaro-7.3.1-2018.05-x86_64_aarch64-linux-gnu.tgz=aarch64-linux-gnu,/home/joke/aarch64-agl-linux-gcc5.3.tar.gz=aarch64-agl-linux"

Weitere Umgebungsvariablen

- Werden mit icecc –help angezeigt

  ICECC if set to "no", just exec the real compiler

  ICECC_VERSION use a specific icecc environment, see icecc-create-env

  ICECC_DEBUG [info | warnings | debug]

  sets verboseness of icecream client.

  ICECC_LOGFILE if set, additional debug information is logged to the specified file

  ICECC_REPEAT_RATE the number of jobs out of 1000 that should be

  compiled on multiple hosts to ensure that they're

  producing the same output. The default is 0.

  ICECC_PREFERRED_HOST overrides scheduler decisions if set.

  ICECC_CC set C compiler name (default gcc).

  ICECC_CXX set C++ compiler name (default g++).

  ICECC_CLANG_REMOTE_CPP set to 1 or 0 to override remote precompiling with clang

  (requires clang -frewrite-includes option).

  ICECC_IGNORE_UNVERIFIED if set, hosts where environment cannot be verified are not used.

  ICECC_EXTRAFILES additional files used in the compilation.
  ICECC_COLOR_DIAGNOSTICS set to 1 or 0 to override color diagnostics support
  ICECC_CARET_WORKAROUND set to 1 or 0 to override gcc show caret workaround

- ICECC_CARET_WORKAROUND: gcc/g++ Auf 0 setzen, um den Caret Workaround abzuschalten. Beschleunigt gerade bei autogenerierten Code den Bau deutlich!

Security

- Praktisch nicht vorhanden
- Benutzt eine Firewall!
- telnet scheduler 8766

  200-ICECC 1.0.1: 8s uptime, 1 hosts, 0 jobs in queue (0 total).
- 200 Use 'help' for help and 'quit' to quit.

  help

  listcs

  listblocks

  listjobs

  removecs

  blockcs

  internals

  help

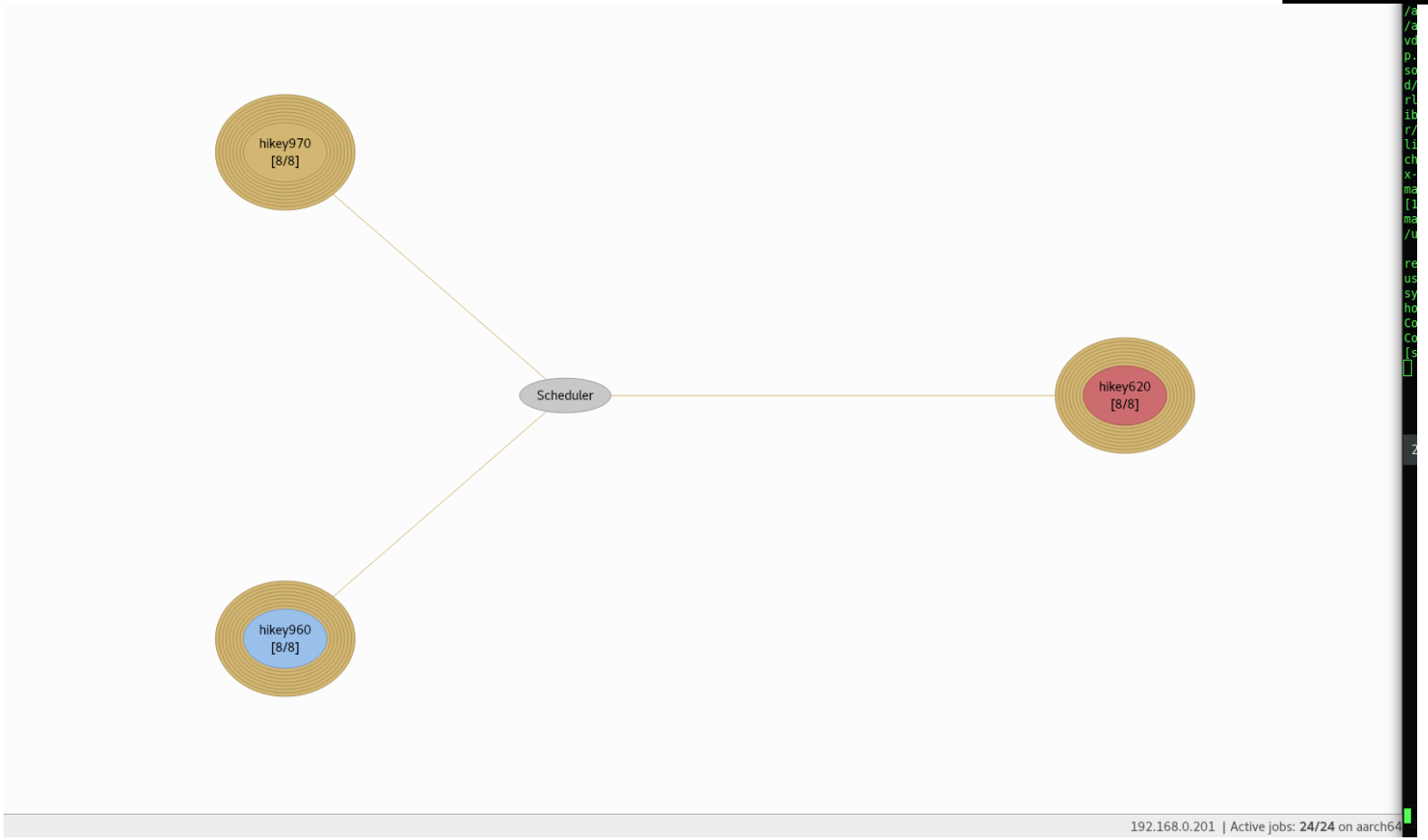
  quit

  200 done
- Blockcs taugt nur um freidrehende/alte Clients loszuwerden, aber dafür könnte man auch ne Firewall nehmen

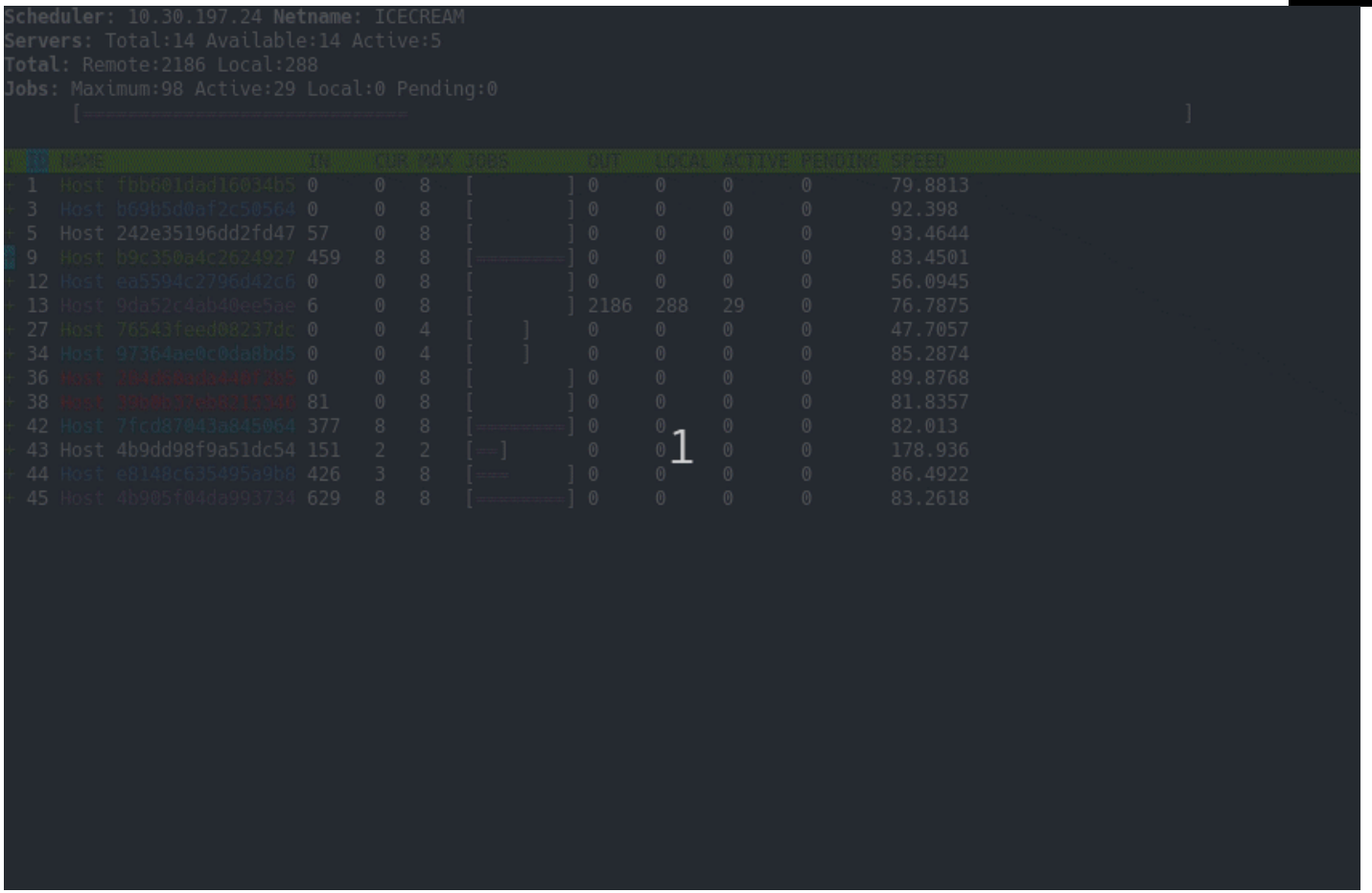
Monitoring

- QT-GUI icemon sudo apt-get install icemon
- icemon -n stratum-cluster
- Icemon -s icecc-scheduler.example
- Curses-GUI: https://github.com/JPEWdev/icecream-sundae.git
- icecream-sundae -n stratum-cluster
  icecream-sundae -s icecc-scheduler.example

Livedemo :)


hikey970
[8/8]
Scheduler
hikey620
[8/8]
hikey960
[8/8]
192.168.0.201 | Active jobs: 24/24 on aarch64

Livedemo :)


Scheduler: 10.30.197.24 Netname: ICECREAM
Servers: Total:14 Available:14 Active:5
Total: Remote:2186 Local:288
Jobs: Maximum:98 Active:29 Local:0 Pending:0
ID NAME IN CUR MAX JOBS OUT LOCAL ACTIVE PENDING SPEED
1 Host fbb601dad16034b5 0 0 8 0 0 0 0 79.8813
3 Host b69b5d0af2c50564 0 0 8 0 0 0 0 92.398
5 Host 242e35196dd2fd47 57 0 8 0 0 0 0 93.4644
9 Host b9c350a4c2624927 459 8 8 0 0 0 0 83.4501
12 Host ea5594c2796d42c6 0 0 8 0 0 0 0 56.0945
13 Host 9da52c4ab40ee5ae 6 0 8 2186 288 29 0 76.7875
27 Host 76543feed08237dc 0 0 4 0 0 0 0 47.7057
34 Host 97364ae0c0da8bd5 0 0 4 0 0 0 0 85.2874
36 Host 2b4d60ada440f2b5 0 0 8 0 0 0 0 89.8768
38 Host 39b0b37e98215348 81 0 8 0 0 0 0 81.8357
42 Host 7fcd87043a845064 377 8 8 0 0 0 0 82.013
43 Host 4b9dd98f9a51dc54 151 2 2 0 0 0 0 178.936
44 Host e8148c635495a9b8 426 3 8 0 0 0 0 86.4922
45 Host 4b905f04da993734 629 8 8 0 0 0 0 83.2618

Icecc mit ccache kombinieren

- Ist es nicht trotzdem Quatsch jedesmal neu zu kompilieren?
- Warum nicht zwischenspeichern?
- ccache hat dafür die Option prefix_command oder Umgebungsvariablen CCACHE_PREFIX:

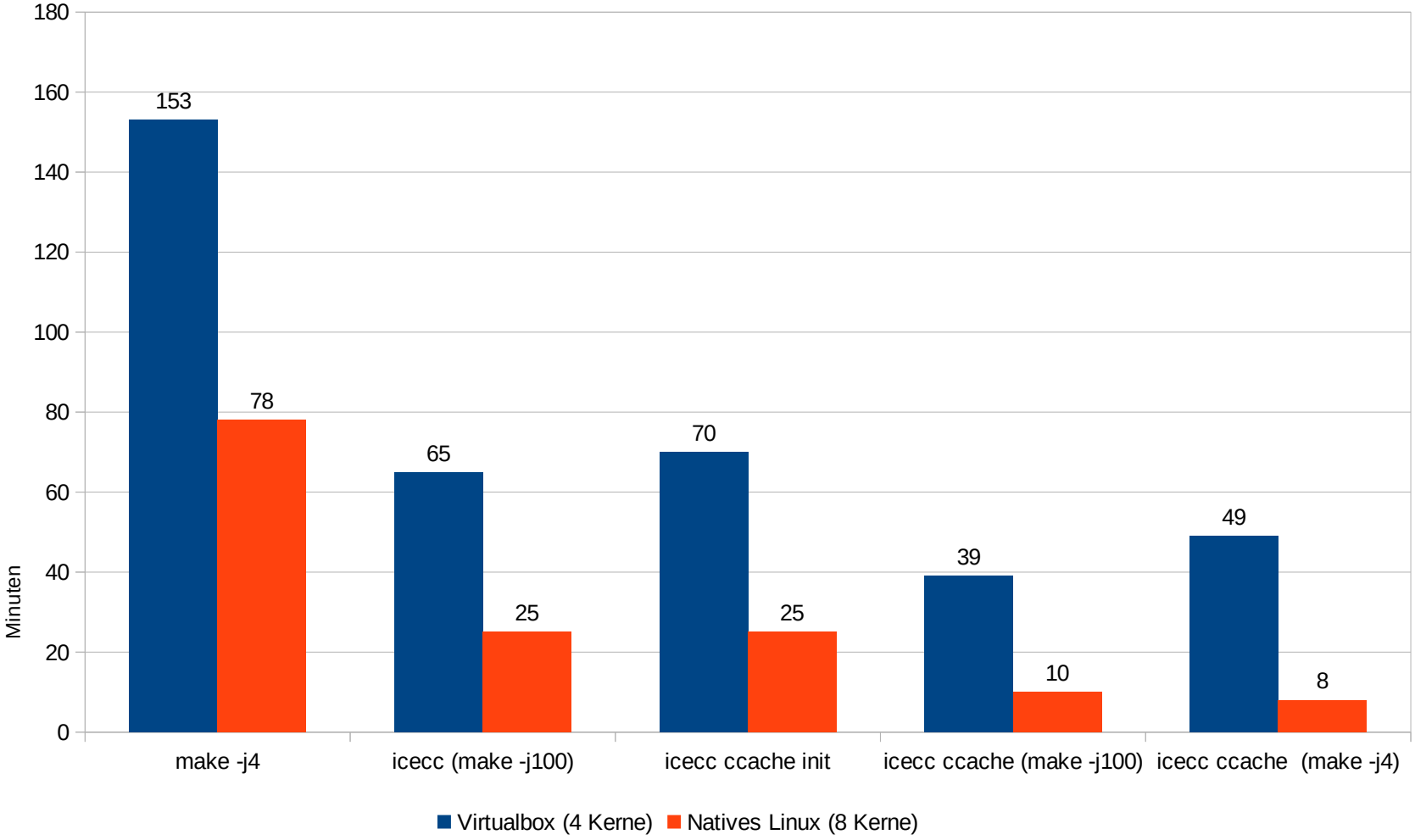
  export CCACHE_PREFIX=icecc #temporär

  ccache -o prefix_command=icecc #dauerhaft
- Unbedingt benchmarken!
- Je nach Szenario unterschiedlich sinnvoll!

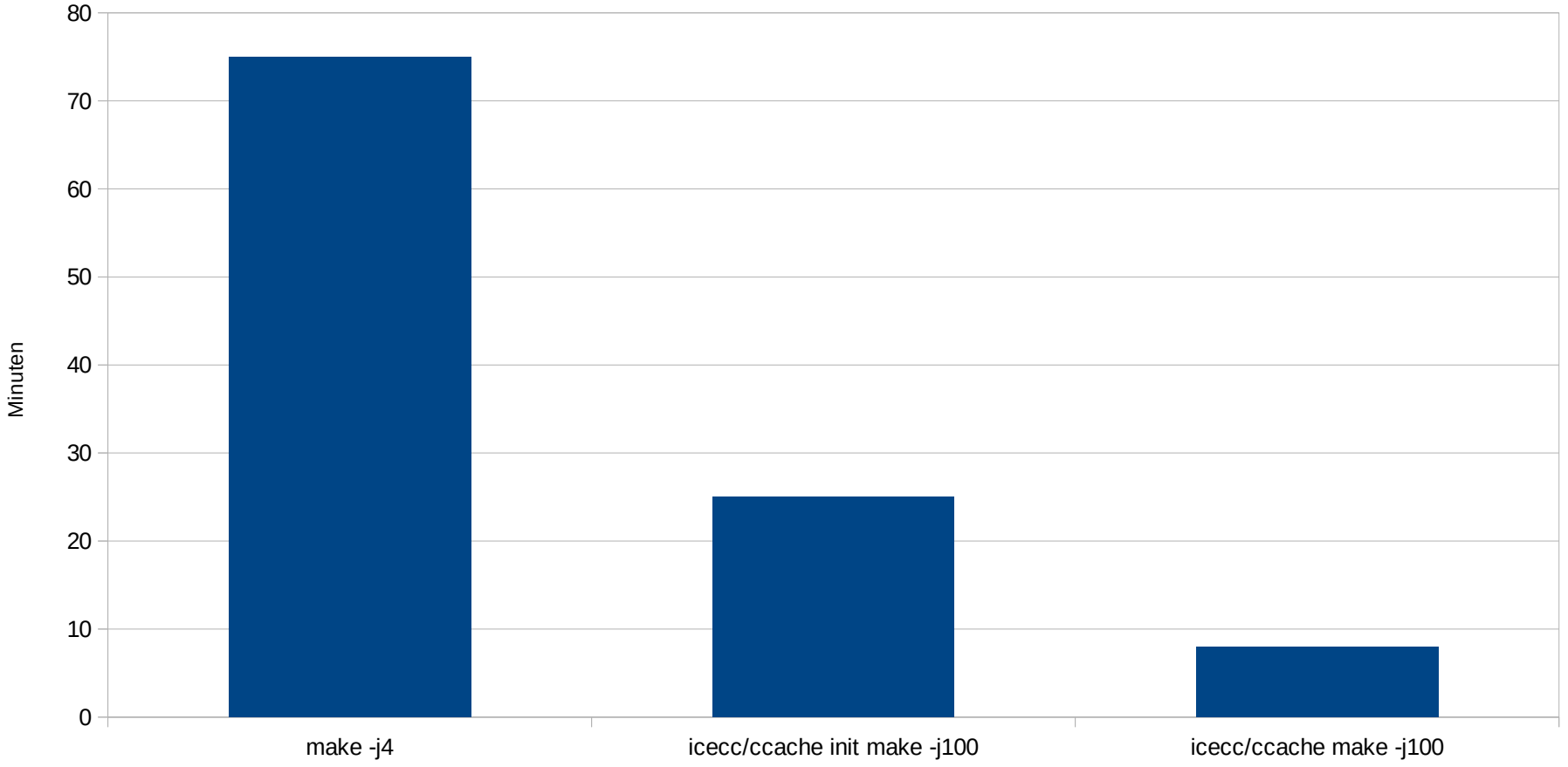
Performance

- Ausgang: Zwei identische Dell-Workstations, je acht Kerne, einmal natives Linux, einmal Windows mit Linux VM
- Linux-VM hat 4 Kerne zugewiesen
- Arbeitsspeicher: Viel (20-30 GB)
- Festplatte: HDD auf nativen Linux, SSD auf VM
- ICECC_CLUSTER mit 120-150 Kernen (je nach Tageszeit) und 20-30 Clients
- Großes Projekt, wird mit eigens entwickelten Buildtool gebaut
- Plattform: x86 für Simulation, arm64 für eigentliche Software

Performance mit Buildtool



Linux Kernel Autoconf/Automake (Virtualbox mit 4 Kernen)



Gibt es das auch für andere Sprachen?

- cache und icecc unterstützt C und C++ und die Apple-Varianten (Objective-C/Objective-C++)
- sccache: Shared Compilation Cache von Mozilla: https://github.com/mozilla/sccache
- Unterstützt rust, C, C++, Nvidia CUDA
- Cached auf lokaler Disk oder cloud storage
- Unterstützt auch verteiltes Compilieren ala icecream/icecc, inklusive Verschlüsselung und Authentifizierung!
- Leider keine eigene Erfahrungen damit

Danke für eure Aufmerksamkeit!

Folien findet ihr mit dem QR-Code oder unter:
https://github.com/johannesst/ccache-icecc-datenburg