Distributed Building

Distributed building can significantly speed up the build process by utilizing multiple machines. However, for ordinary NixOS users, distributed building may not be very useful since `cache.nixos.org` provides a vast majority of caches for the `x86_64` architecture.

Distributed building is particularly valuable in scenarios where no cache is available, such as:

1. Users of `RISC-V` or `ARM64` architectures, especially `RISC-V`, as there are very few caches for these architectures in the official cache repository. Local compilation is often required.
2. Users who heavily customize their systems. The packages in the official cache repository are built with default configurations. If you modify the build parameters, the official cache is not applicable, and local compilation is necessary. For example, in embedded scenarios, customization of the underlying kernel, drivers, etc., is often required, leading to the need for local compilation.

Configuring Distributed Building

Currently, there is no official documentation for distributed building. However, I have provided a sample distributed build configuration (a NixOS module) below, along with some recommended reference documents at the end of this section.

nix

```nix
{ ... }: {

  ######################################################################
  #
  #  NixOS's Configuration for Remote Building / Distributed Building
  #
  ######################################################################

  # Set local's max-jobs to 0 to force remote building (disable local building).
  # nix.settings.max-jobs = 0;
  nix.distributedBuilds = true;
  nix.buildMachines =
    let
      sshUser = "ryan";
```

```
        # Path to the SSH key on the local machine.
        sshKey = "/home/ryan/.ssh/ai-idols";
        systems = [
          # Native architecture.
          "x86_64-linux"

          # Emulated architecture using binfmt_misc and qemu-user.
          "aarch64-linux"
          "riscv64-linux"
        ];
        # All available system features are poorly documented here:
        # https://github.com/NixOS/nix/blob/e503ead/src/libstore/globals.hh#L673-L
        supportedFeatures = [
          "benchmark"
          "big-parallel"
          "kvm"
        ];
      in
        [
          # Nix seems to always prioritize remote building.
          # To make use of the local machine's high-performance CPU, do not set th
          {
            # Some of my remote builders are running NixOS
            # and have the same sshUser, sshKey, systems, etc.
            inherit sshUser sshKey systems supportedFeatures;

            # The hostName should be:
            #   1. A hostname that can be resolved by DNS.
            #   2. The IP address of the remote builder.
            #   3. A host alias defined globally in /etc/ssh/ssh_config.
            hostName = "aquamarine";
            # Remote builder's max-jobs.
            maxJobs = 3;
            # SpeedFactor is a signed integer,
            # but it seems that it's not used by Nix and has no effect.
            speedFactor = 1;
          }
          {
            inherit sshUser sshKey systems supportedFeatures;
            hostName = "ruby";
            maxJobs = 2;
            speedFactor = 1;
          }
          {
```

```
        inherit sshUser sshKey systems supportedFeatures;
        hostName = "kana";
        maxJobs = 2;
        speedFactor = 1;
      }
    ];
  # Optional: Useful when the builder has a faster internet connection than your
    nix.extraOptions = ''
      builders-use-substitutes = true
    '';

  # Define the host aliases for remote builders.
  # This configuration will be written to /etc/ssh/ssh_config.
  programs.ssh.extraConfig = ''
    Host ai
      HostName 192.168.5.100
      Port 22

    Host aquamarine
      HostName 192.168.5.101
      Port 22

    Host ruby
      HostName 192.168.5.102
      Port 22

    Host kana
      HostName 192.168.5.103
      Port 22
  '';

  # Define the host keys for remote builders so that Nix can verify all the remo
  # This configuration will be written to /etc/ssh/ssh_known_hosts.
  programs.ssh.knownHosts = {
    # 星野 愛久愛海, Hoshino Aquamarine
    aquamarine = {
      hostNames = [ "aquamarine" "192.168.5.101" ];
      publicKey = "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDnCQXlllHoLX5EvU+t6yP/np
    };

    # 星野 瑠美衣, Hoshino Rubii
    ruby = {
      hostNames = [ "ruby" "192.168.5.102" ];
      publicKey = "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIE7n11XxB8B3HjdyAsL3PuLVD
```

```
    };

    # 有馬 かな, Arima Kana
    kana = {
      hostNames = [ "kana" "192.168.5.103" ];
      publicKey = "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIJ3dDLOZERP1nZfRz3zIeVDm1
    };
  };
}
```

Limitations

Here are some observed issues and limitations:

1. You cannot specify which hosts to use at build time. You can only specify a list of hosts in the configuration file, and Nix automatically selects available hosts.
2. When choosing a host, Nix always prefers the remote host over the local host, even if the local host has better performance. This can result in underutilization of the local host's CPU.
3. The smallest unit of distributed building is a derivation. When building large packages, other machines may remain idle for a long time, waiting for the large package to be built. This can lead to resource wastage.

References

- Distributed build - NixOS Wiki
- Document available system features - nix#7380
- Distributed builds seem to disable local builds - nix#2589
- Offloading NixOS builds to a faster machine
- tests/nixos/remote-builds.nix - Nix Source Code

Loading comments...