PROJEKT Z PRZEDMIOTU PRZETWARZANIE ROZPROSZONE

Autor: Bartłomiej Rosa

Temat projektu

Równoległa implementacja algorytmu sortowania przez scalanie (mergesort).

Opis algorytmu

Sortowanie przez scalanie osiąga złożoność czasową O*(n*log(n)) i pamięciową O(n*) zarówno w przypadku optymistycznym, średnim i pesymistycznym. Ideą działania algorytmu jest dzielenie zbioru danych na mniejsze zbiory, aż do uzyskania *n* zbiorów jednoelementowych (zbiór jednoelementowy jest z zawsze posortowany). Posortowane następnie zbiory te są łączone w coraz większe zbiory posortowane, aż do uzyskania jednego, posortowanego zbioru *n* elementowego.

Implementacja równoległa

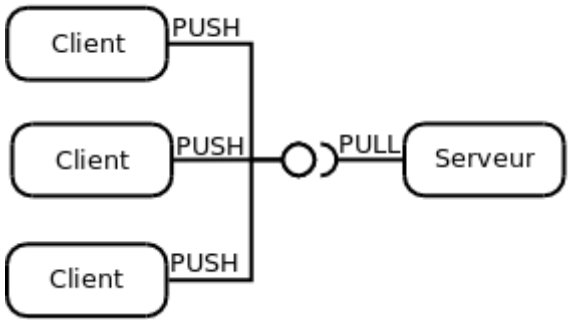
Plik *Czesc1.py* zawiera metodę główną, której zadaniem jest odczytanie od użytkownika rozmiaru tablicy do posortowania, która zostania wygenerowana. Użytkownik podaje też maksymalną liczbę procesów, które mają zostać wykorzystane. Do posortowania tablicy rozmiaru *n* wystarczy co najwyżej *log(n)+1* procesów. Plik *MergeSortAlgorithm.py* to implementacja algorytmu (funkcje merge, mergesort). Plik *MergeSortMultiprocess.py* zawiera metodę *testMergeSort* tworzącą proces z metody *mergeSortProcess*, która to metoda tworzy kolejne procesy jeśli jest taka potrzeba.

Zdecydowałem się na użycie w projekcie biblioteki Multiprocessing zamiast Threading przede wszystkim dlatego, że jest w dla wątków jest używany Global Interpreter Lock, który nie pozwala dwóm wątkom działać w tym samym czasie. Procesy są też bardziej odseparowane ze względu na działanie na oddzielnej przestrzenie adresowej. Minusem jest, że wstawanie procesów jest wolniejsze.

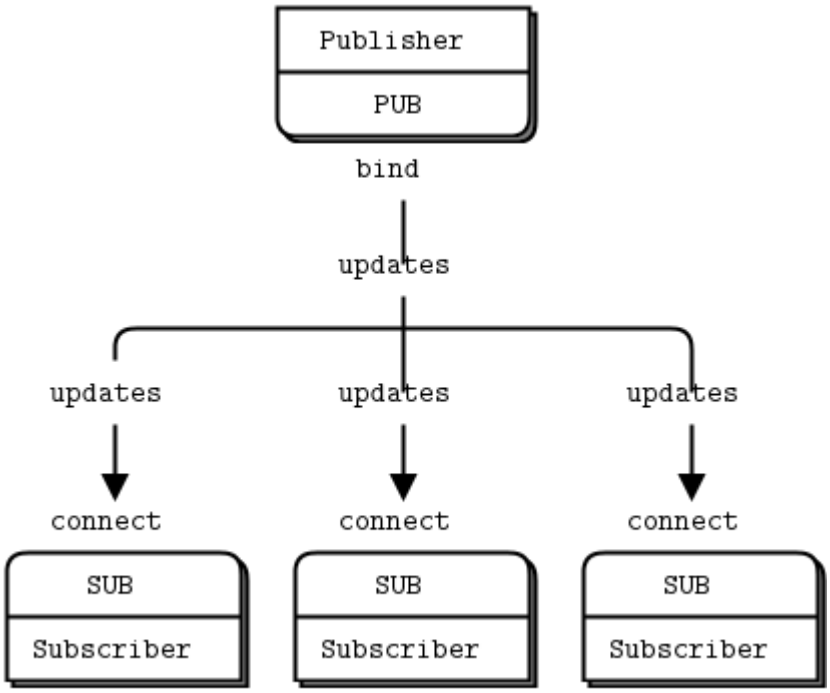
Implementacja rozproszona

Pliki *MasterWorkerRunner.py* oraz *SlaveWorkerRunner.py* to pliki uruchomieniowe węzła master i slave w implementacji rozproszonej. Do uruchomienia środowiska służy skrypt *runExample*. Pliki *MasterWorker.py* oraz *SlaveWorker.py* zawierają kod węzłów. Każdy węzeł jest określany jako tzw. worker.

W swojej implementacji zastosowałem bibliotekę ZeroMQ, która ułatwia korzystanie z pewnych wzorców komunikacyjnych w sieci. Do komunikacji mastera ze slave wykorzystałem wzorzec push/pull. Jest to przedstawione na rysunku. W moim przypadku Producer oraz result collector to ten sam węzeł.



Do informowania węzłów slave o zakończeniu działania użyłem wzorzec publish/subscribe. W tym wzorcu istnieje jedeno gniazo sieciowe PUB (od publish) oraz zero lub więcej gniazd sieciowych SUB (od subscribe). Gniazdo PUB rozsyła do wszystkich gniazd informację o zakończeniu obliczeń.



Bibliografia

1. Dokumentacja multiprocessing, https://docs.python.org/2/library/multiprocessing.html
2. Dokumentacja biblioteki ZeroMQ, http://zeromq.org/whitepapers:brokerless