Decentralizacja sterowania robota mobilnego przy wykorzystaniu klastra komputerowego

Michał Podpora

***Streszczenie*— W artykule autor przedstawia zalety i wady wykorzystania klastra komputerowego jako sposobu na decentralizację mocy obliczeniowej i sterowania robota mobilnego. W przytoczonej przykładowej realizacji praktycznej priorytetem jest przeniesienie na klaster mechanizmów sztucznej inteligencji i sterowania, celem odciążenia jednostki bazowej wyposażonej m.in. w kamerę cyfrową.**

***Słowa kluczowe*— Klaster, Sterowanie, Systemy wizyjne.**

I. Wstęp

Od wielu robotów mobilnych wymaga się zarówno małych rozmiarów jak i dużych możliwości. Trudno wyobrazić sobie maleńkiego robocika wożącego ze sobą potężny superkomputer. Okazuje się jednak, że możliwe jest wyposażenie konstruowanego „w domowych warunkach" urządzenia w ogromną moc obliczeniową i to nie zwiększając ani jego masy ani wymiarów. Decentralizacja mocy obliczeniowej uzyskiwana jest poprzez przeniesienie części algorytmu na inny komputer połączony bezprzewodowo do sterowanego urządzenia. Może być to osiągnięte poprzez zastosowanie dwóch komputerów i aplikacji sterującej o architekturze klient-serwer, lub poprzez zastosowanie klastra. Tak w pierwszym jak i w drugim przypadku robot może być wyposażony np. w beznapędowy lekki laptop, podczas gdy algorytm sterowania może być zaimplementowany na innym komputerze.

II. Dlaczego klaster

Przeniesienie części algorytmu na drugi komputer jest rozwiązaniem prostym i skutecznym. Po co się więc trudzić z klastrem? Każdy nawet początkujący programista potrafi stworzyć aplikację typu klient-serwer, a użycie dwóch komputerów zazwyczaj wystarcza: jeden tani lekki laptop do agregacji danych pozyskiwanych z wejść i do sterowania mechaniką oraz drugi komputer – potężny, stacjonarny, zajmujący się przetwarzaniem pozyskanych przez laptop danych, analizowaniem i wnioskowaniem, a w efekcie podejmowaniem decyzji co do sterowania robotem.

Wspomniana architektura niejednokrotnie jest w zupełności wystarczająca, zdarzają się jednak bardziej wymagające implementacje. Czasem pomimo zastosowania komputera o wysokich parametrach, system nie nadąża z reagowaniem na pojawiające się sytuacje. Jeżeli robot ma „zdążyć na czas" z zareagowaniem (ponieważ np. przed nim są schody/ściana), to jego reakcja musi nastąpić wystarczająco wcześnie. Jeżeli dodatkowo korzysta z kamery jako jednego z wejść informacji o otoczeniu, to ważne jest, aby algorytm decyzyjny pracował na możliwie najbardziej aktualnym fragmencie sekwencji wideo. Gdyby analiza jednej klatki tej sekwencji zajmowała kilka sekund, to po tym czasie robot może znajdować się w diametralnie odmiennym położeniu.

W takiej sytuacji, gdy spodziewamy się ogromnego zapotrzebowania na moc obliczeniową, a w dodatku istnieje potrzeba ukończenia analizy każdego fragmentu danych w jak najkrótszym czasie, warto zastanowić się nad użyciem klastra komputerowego.

III. Dobór odpowiedniej topologii klastra

Po wstępnym skonfigurowaniu hostów (/etc/hosts, firewall, ssh, lam) w czym w razie problemów mogą okazać się pomocne zasoby Internetu [2], [3], projektant programista staje przed problemem doboru odpowiedniej topologii klastra. Z założenia jednym z węzłów klastra jest wcześniej wymieniony laptop zbierający informacje o otoczeniu i sterujący robotem. Korzystając z faktu, że uszkodzenie tego komputera i tak unieruchomiłoby cały system, można go uczynić „sercem" całej aplikacji (dodatkowo pełni wówczas funkcje koordynacji pomiędzy węzłami klastra). Reasumując: jasno określone jest wejście danych i wyjście sterowania, a statyczne predefiniowanie koordynatora dodatkowo upraszcza algorytm. Jeżeli weźmiemy pod uwagę wspomniane warunki, decyzja co do wyboru rodzaju topologii staje się łatwiejsza.

A. Pierścień

Ponieważ zarówno wejście jak i wyjście znajduje się na tym samym węźle, topologia pierścienia (1) wydaje się być warta omówienia.

Niestety sposób przetwarzania danych praktycznie wyklucza tą topologię – za każdym razem przetwarzana jest jedna (kolejna) klatka sekwencji wideo, co więcej algorytm powinien ukończyć przetwarzanie danej klatki zanim pojawi się następna. Taka sytuacja powoduję, że węzły klastra o topologii pierścienia nie mogą równocześnie przetwarzać kolejnych porcji danych w efekcie funkcjonując po prostu sekwencyjnie. Ponadto użycie topologii pierścienia wymusza

użycie algorytmu synchronicznego, podczas gdy algorytm asynchroniczny redukuje czas oczekiwania [1].

B. *Gwiazda*

Topologia pierścienia w ogóle nie korzystała z faktu, że węzeł wejściowy może pełnić równocześnie rolę koordynatora. Za to topologia gwiazdy (1) wydaje się spełniać wszystkie nasze wymagania. Wystarczy tylko rozbić ciężar obliczeń na tyle niezależnych fragmentów, ile jest dodatkowych węzłów.

Niestety topologia gwiazdy ma jeden ogromny mankament – a jest nim komunikacja. Pozyskane klatki sekwencji wideo (lub ich fragmenty) muszą być porozsyłane do praktycznie wszystkich węzłów. Czas przesyłania danych przy użyciu sieci komputerowej pomiędzy węzłami jest bez porównania większy od czasu przetwarzania, dlatego w naszym specyficznym przypadku gwiazda nie jest dobrym wyborem.

C. *Drzewo*

Topologia drzewa (1) jest technicznie bardziej skomplikowana do zaimplementowania niż pierścień czy gwiazda, jednakowoż trud włożony w aplikację przynosi wymierne efekty. Węzeł wejściowy przesyła dane wejściowe do kilku (nie wszystkich) węzłów. Te węzły wykonują część przekształceń i przekazują dane kolejnym węzłom. Węzły na kolejnych poziomach mogą być dobrane tak, aby nie było potrzeby przesyłania kompletnych danych. Załóżmy, że dany węzeł zajmuje się znajdowaniem konturów i identyfikacją/ /klasyfikacją kształtów. Wynik jego działania może być przekazywany do kolejnego węzła (jeszcze jeden poziom dalej względem węzła wejściowego), który podejmie decyzję co do identyfikacji/klasyfikacji obiektów w oparciu o wykryte kształty.

Przy topologii gwiazdy należało rozbić ciężar obliczeń, natomiast w przypadku drzewa lepiej jest brać pod uwagę czas komunikacji.

D. *Drzewo „z poprawkami”*

Klasyczna koncepcja drzewa jest nieco zbyt sztywna dla naszej implementacji – nie jest możliwy dobry rozdział zadań poszczególnym węzłom; nie wszystkie zadania wymagają/ /umożliwiają podział na dwa lub więcej fragmentów które mogłyby być realizowane przez dwa węzły z sąsiednich poziomów drzewa.

Ostatni szkic na rysunku (1) symbolizuje taki właśnie przypadek. Węzeł wejściowy przesyła wybrane dane niektórym węzłom, które następnie analizują otrzymane dane pod kątem specyficznej cechy/stanu [4] lub też po prostu przetwarzają przygotowując dla następnych węzłów.

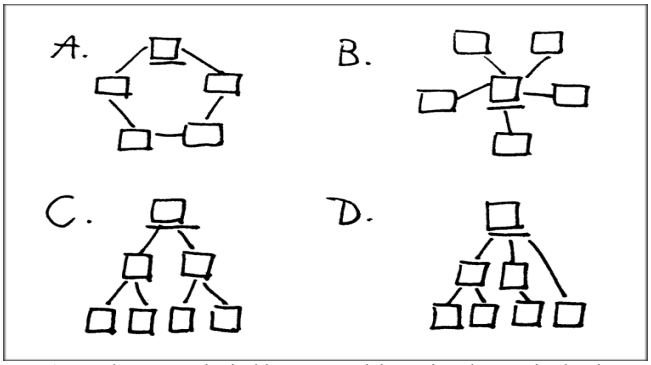
IV. Realizacja praktyczna i wnioski

Aplikacja [4] tworzona na potrzeby moich badań pokazuje, że rozdzielanie zadań, jeżeli czynione rozważnie, pozwala na osiągnięcie zadowalających wyników. Biorąc pod uwagę hybrydową topologię tworzoną na bazie topologii drzewa, o skalowalnym [1] fragmencie zawierającym algorytmy sieci neuronowych, możliwości tej aplikacji rysują się obiecująco. Fizyczna rozbudowa klastra nie zmienia tu podstawowego podziału zadań (rozpoznawanie ruchu, kolorów, kształtów), natomiast teoretycznie powinna zwiększać możliwości sieci neuronowych zaimplementowanych w aplikacji (2).

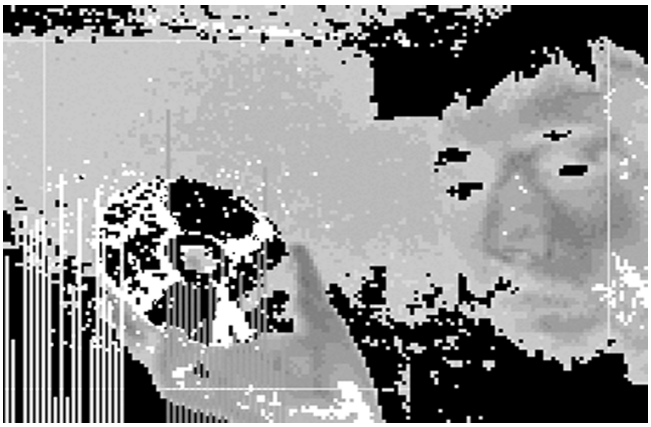
Aplikacja (2) jest aktualnie w początkowej fazie tworzenia.

Bibliografia

[1] A. Karbowski, E. Niewiadomska-Szynkiewicz, *Obliczenia równoległe i rozproszone*, Oficyna Wydawnicza Politechniki Warszawskiej, (2001)
[2] LAM-MPI user's mailing lists, dostępny: http://www.lam-mpi.org
[3] MPI standards, dostępny: http://www.mpi-forum.org/docs/docs.html
[4] M. Podpora, *Computer Vision in Parallel Computing*, ISTET'07 - materiały konferencyjne, przyjęte do publikacji



Rys. 1. Wybrane topologie klastrów. Kolejno: pierścień, gwiazda, drzewo, „drzewo z poprawkami” – omówione w rozdziale III.



Rys. 2. Podgląd procesów decyzyjnych realizacji praktycznej – aplikacji [4]. Histogramy nałożone na obraz reprezentują odpowiedzi sieci neuronowych z wybranych węzłów (lepiej widoczne w kolorze); nasycenie barw oznacza próg zainteresowania kolorem obiektu; nie jest natomiast widoczny próg zainteresowania ruchem oraz kilka innych czynników

Abstract— In the paper author describes advantages and disadvantages of using a computer cluster to decentralize computation and control algorithms of a mobile robot. In a practical realization author uses the cluster as a center for artificial intelligence and steering – to speed up the input node with a digital camera.

Topic— Decentralization of control of a mobile robot by use of a computer cluster