

Úvod do distribuovaných výpočtů

18. dubna 2018

B4B36PDV – Paralelní a distribuované výpočty

- Distribuovaný výpočet
- DSand framework
- Distribuované prohledávání

Vyžadujeme samostatnou práci na všech úlohách.

⚠ Plagiáty jsou zakázané. Nepřidělávejte prosím starosti nám, ani sobě.

```

}
}

void radix_sample(std::vector<std::string *> &vector_to_sort, const MappingF
                unsigned long alphabet_size, unsigned long string_lengths,
                std::vector<std::string *> buckets[alphabet_size];

for (long i = string_lengths - 1; i >= 1; --i) {
    for (long j = from; j < to; ++j) {
        std::string *str = vector_to_sort[j];
        unsigned long pos = mappingFunction((*str).at(i));
        buckets[pos].push_back(str);
        //std::cout << pos << " " << *str << std::endl;
    }
    //std::cout << "END" << std::endl;

    //
    long counter = from;
    for (int k = 0; k < alphabet_size; ++k) {
        long ff = buckets[k].size();
        for (int j = 0; j < ff; ++j) {
            vector_to_sort[counter] = buckets[k][j];
            counter++;
        }
        buckets[k].clear();
    }
}
}
}

```

```

                vector_to_sort.push_back(k);
    }
    end += buckets[cnt].size();
    #pragma omp task
    inside_func(vector_to_sort, mappingFunction, alphabet_size,
    //
    begin += buckets[cnt].size();
    buckets[cnt].clear();
    ++cnt;
}
}
}

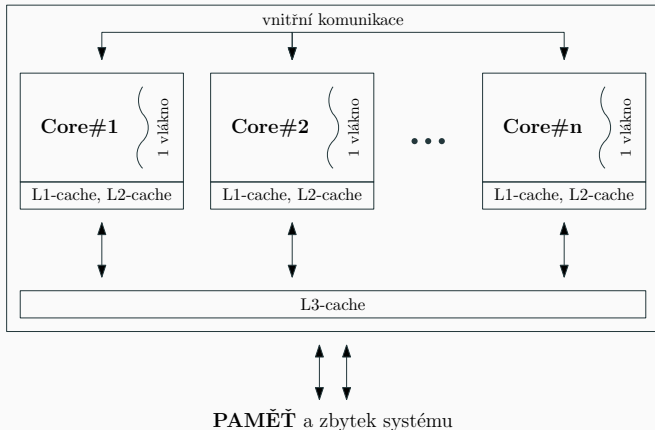
void inside_func(std::vector<std::string *> &vector_to_sort, const MappingF
                std::vector<std::string *> buckets[alphabet_size];
for (long i = string_lengths - 1; i >= 1; --i) {
    for (long j = begin; j < end; ++j) {
        std::string *str = vector_to_sort[j];
        unsigned long position = mappingFunction((*str).at(i));
        buckets[position].push_back(str);
    }
    long cnt = begin;
    for (int k = 0; k < alphabet_size; ++k) {
        long ff = buckets[k].size();
        for (int j = 0; j < ff; ++j) {
            vector_to_sort[cnt] = buckets[k][j];
            ++cnt;
        }
        buckets[k].clear();
    }
}
}
}
}

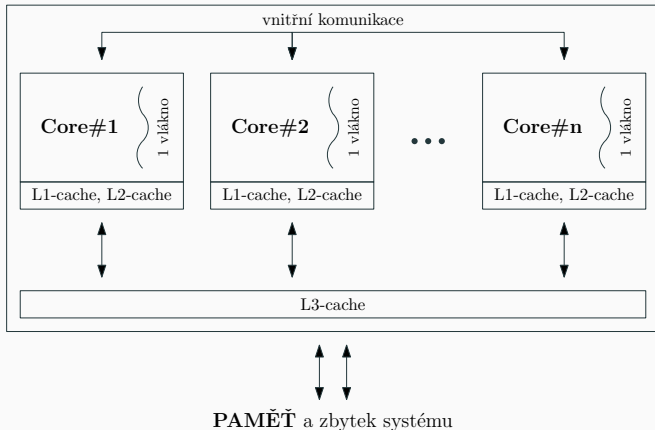
```

⚠ Pokud se poznáváte (respektive víte, že jste na některé úloze spolupracovali s kolegou v míře větší než malé), tak se nám raději ozvěte včas!

Distribuovaný výpočet

Moderní procesor





Proč bychom mohli chtít tento model opustit?

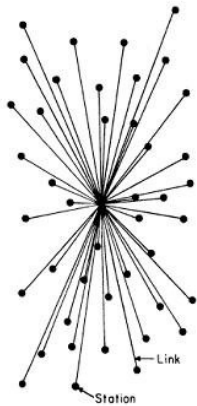
- I ty nejšílenější procesory mají omezený počet vláken (za šílenou cenu)
(Xeon Platinum 8180, 56 vláken, ~\$10000)

- I ty nejšílenější procesory mají omezený počet vláken (za šílenou cenu)
(Xeon Platinum 8180, 56 vláken, ~\$10000)
- Přístupy do sdílené paměti jsou drahé
(I pokud nepotřebujeme synchronizovat! – sběrnice má omezenou kapacitu)

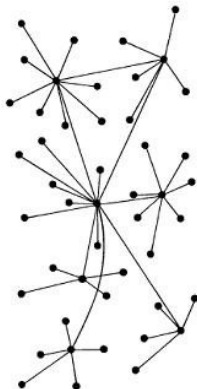
- I ty nejšílenější procesory mají omezený počet vláken (za šílenou cenu)
(Xeon Platinum 8180, 56 vláken, ~\$10000)
- Přístupy do sdílené paměti jsou drahé
(I pokud nepotřebujeme synchronizovat! – sběrnice má omezenou kapacitu)
- Jediný „stroj“ je *single point of failure*
(Pokud nám selže, tak jsme v háji)

- I ty nejšílenější procesory mají omezený počet vláken (za šílenou cenu)
(Xeon Platinum 8180, 56 vláken, ~\$10000)
- Přístupy do sdílené paměti jsou drahé
(I pokud nepotřebujeme synchronizovat! – sběrnice má omezenou kapacitu)
- Jediný „stroj“ je *single point of failure*
(Pokud nám selže, tak jsme v háji)
- Chceme být geograficky blíže cílovým uživatelům
(Nižší latence, ...)

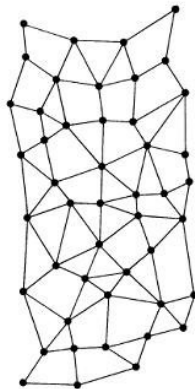
- I ty nejšílenější procesory mají omezený počet vláken (za šílenou cenu)
(Xeon Platinum 8180, 56 vláken, ~\$10000)
- Přístupy do sdílené paměti jsou drahé
(I pokud nepotřebujeme synchronizovat! – sběrnice má omezenou kapacitu)
- Jediný „stroj“ je *single point of failure*
(Pokud nám selže, tak jsme v háji)
- Chceme být geograficky blíže cílovým uživatelům
(Nižší latence, ...)
- „Důvěryhodnost“ výpočtu
(Pokud nám výpočet zverifikuje několik *nezávislých* strojů, je pravděpodobně správný)



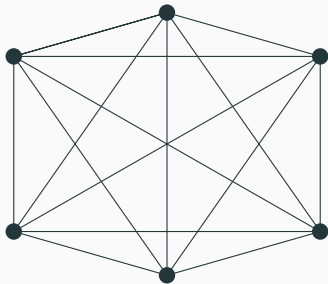
CENTRALIZED
(A)



DECENTRALIZED
(B)



DISTRIBUTED
(C)



Na jaké problémy ale narazíme?

Na jaké problémy ale narazíme?

- Nespolehlivá komunikace (ztracené/zpožděné zprávy)
- Nespolehlivé výpočetní jednotky (musíme počítat se selháním procesu)

Na jaké problémy ale narazíme?

- Nespolehlivá komunikace (ztracené/zpožděné zprávy)
- Nespolehlivé výpočetní jednotky (musíme počítat se selháním procesu)

A v nejhorším případě až:

- Zprávy doručené v chybném pořadí (UDP vs. TCP)
- Poškozené zprávy
- Procesy, které se nás snaží cíleně podvést (například, infikované uzly)

Čím se budeme zabývat?

- Výpočet provádí současně více oddělených výpočetních uzlů (často i geograficky – my budeme ale distribuovanost simulovat)
- Cíle:
 - Zrychlit výpočet
 - Robustnost výpočtu
- (5 týdnů)

Úlohy z distribuované části (min. 9 bodů)

- | | |
|---------------------|---------|
| ▪ 2 malé úlohy | 4 body |
| ▪ Semestrální práce | 14 bodů |

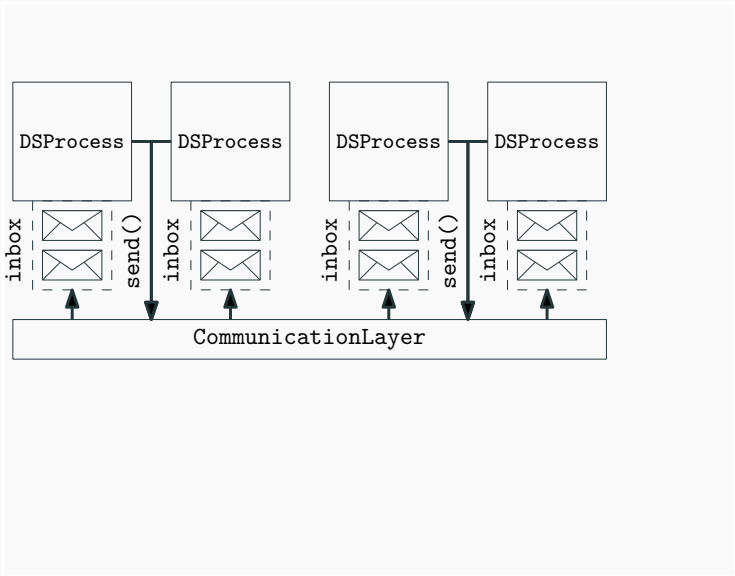
Čím se budeme zabývat?

- Výpočet provádí současně více oddělených výpočetních uzlů (často i geograficky – my budeme ale distribuovanost simulovat)
- Cíle:
 - Zrychlit výpočet
 - **Robustnost výpočtu**
- (5 týdnů)

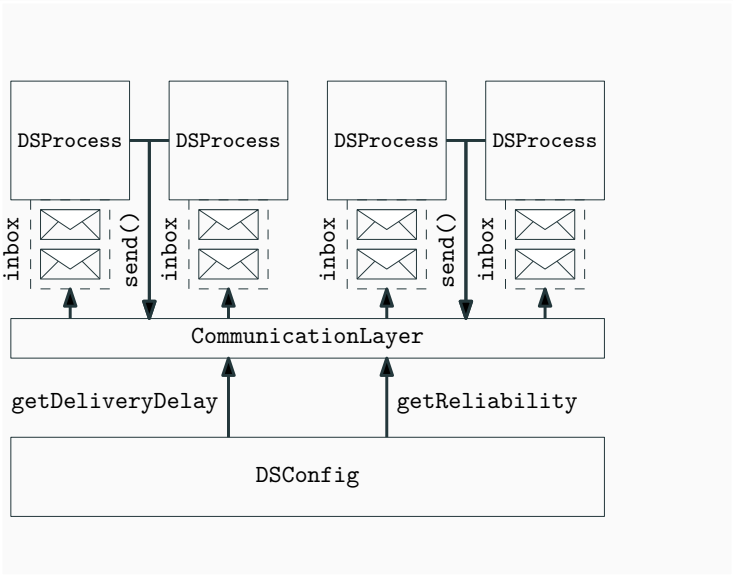
Úlohy z distribuované části (min. 9 bodů)

- | | |
|---------------------|---------|
| ▪ 2 malé úlohy | 4 body |
| ▪ Semestrální práce | 14 bodů |

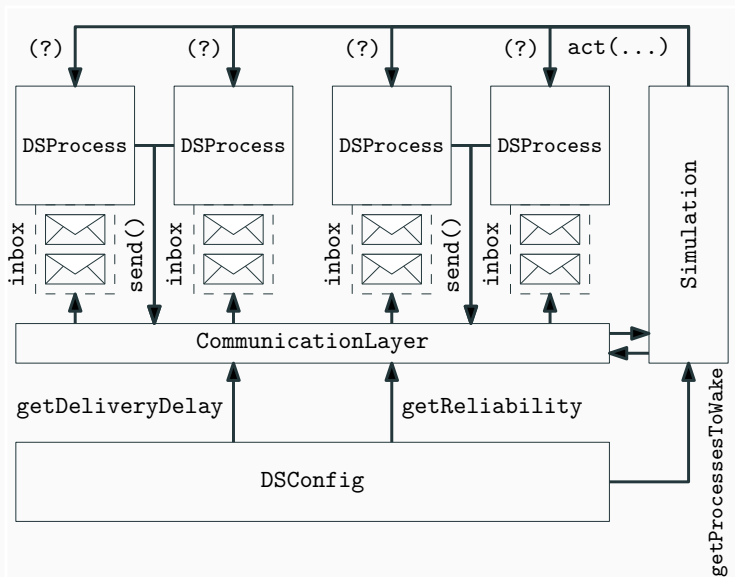
DSand framework



DSand framework



DSand framework



Seznamte se s frameworkem DSand

Spusťte hlavní třídu `demo.Main` a vyzkoušejte si framework DSand. Můžete si i zkusit změnit parametry systému ve třídě `demo.DSConfig` a pozorovat, jaký vliv tato změna má na chování distribuovaného systému.

Synchronní distribuovaný systém

Uvažujme na začátek ideální distribuovaný systém...

- Zprávy se nám **neztrácí** a jsou doručované po konstantní době
- Procesy nehavarují a pracují **synchronně** (probouzí se všichni naráz)

Uvažujme na začátek ideální distribuovaný systém...

- Zprávy se nám **neztrácí** a jsou doručované po konstantní době
- Procesy nehavarují a pracují **synchronně** (probouzí se všichni naráz)

To je ještě optimističtější předpoklad,
než jsme měli v paralelizaci!

Hledáme nejkratší cestu do cíle.

- Stavový prostor je silně souvislý
- Neznáme dopředu velikost ani strukturu prostoru

Hledáme nejkratší cestu do cíle.

- Stavový prostor je silně souvislý
 - Neznáme dopředu velikost ani strukturu prostoru
-

Jak namapujeme stavový prostor na procesy v síti?

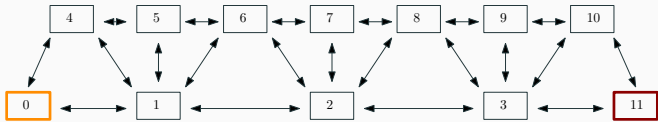
Hledáme nejkratší cestu do cíle.

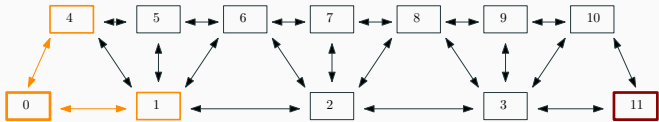
- Stavový prostor je silně souvislý
 - Neznáme dopředu velikost ani strukturu prostoru
-

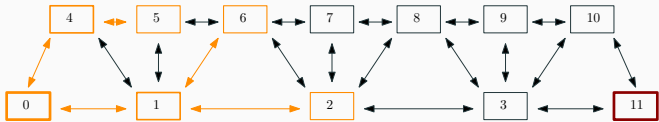
Jak namapujeme stavový prostor na procesy v síti?

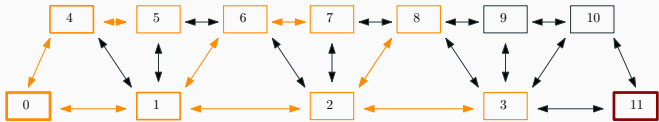
- Každý stav je **samostatný proces**.
- Každý proces zná jen své následovníky.
- Jeden z procesů je **kořen**.

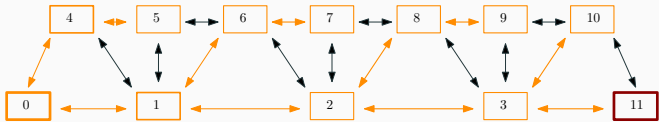
Kořen lze zvolit pomocí volby leadera!











Jaký algoritmus bude provádět každý proces?

Jaký algoritmus bude provádět každý proces?

Označujeme procesy, když jsou **poprvé navštíveny**

- Na začátku je označený jen kořen
- Kořen pošle zprávu `hledej` všem svým následovníkům

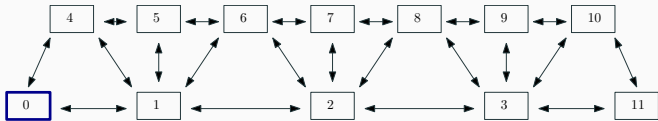
Jaký algoritmus bude provádět každý proces?

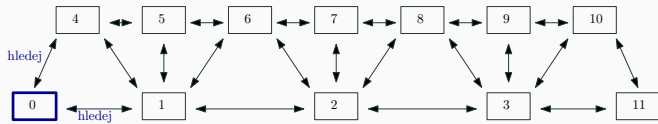
Označujeme procesy, když jsou **poprvé navštíveny**

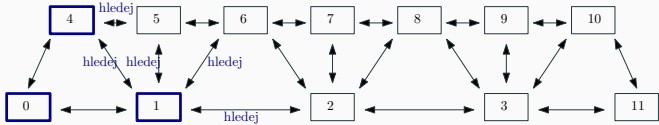
- Na začátku je označený jen kořen
- Kořen pošle zprávu *hledej* všem svým následovníkům

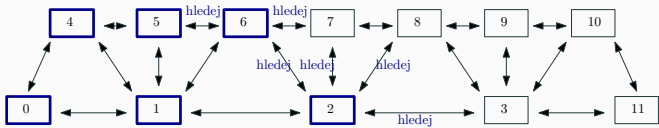
V každém kole *neoznačené* procesy, které přijmou zprávu *hledej*, provedou

- Označí se
- Určí jednoho z procesů, od kterého přijal zprávu, jako předka
- Pošlou zprávu *hledej* všem svým následovníkům









Naimplementujte distribuované prohledávání do šířky

Do souboru `BFSPProcess.java` doplňte kód procesů, které budou prohledávat síť do šířky. Předpokládejte, že systém je synchronní. Ve chvíli kdy najdete cíl, ukončete celý distribuovaný výpočet pomocí metody `terminateAll(...)`.

Jak vypsát posloupnost procesů na cestě k cíli?

Jak vypsát posloupnost procesů na cestě k cíli?

Propagujeme informaci napříč sítí!

Doimplementujte konstrukci cesty z cíle do kořene

Přidejte do Vašeho algoritmu pro distribuované prohledávání zpětnou rekonstrukci cesty z cíle do kořene. Ve chvíli kdy kořen bude znát celou cestu, vypište ji a ukončete distribuovaný výpočet.

Jak bude tento algoritmus fungovat, když systém nebude synchronní?

Jak bude tento algoritmus fungovat, když systém nebude synchronní?

Vyzkoušejte nesynchronní scénáře

Zkuste spustit scénáře `SlowOptimalPath.java`, ve kterém jsou zprávy pouze po optimální cestě zpožděny a `FailingNode2.java`, kdy se zprávy procesu 2 ztrácí. Jaké jsou výsledky jednotlivých scénářů?

Jak bude tento algoritmus fungovat, když systém nebude synchronní?

Vyzkoušejte nesynchronní scénáře

Zkuste spustit scénáře `SlowOptimalPath.java`, ve kterém jsou zprávy pouze po optimální cestě zpožděny a `FailingNode2.java`, kdy se zprávy procesu 2 ztrácí. Jaké jsou výsledky jednotlivých scénářů?

⚠ Nemáme žádné garance. Výsledek nemusí být optimální, dokonce ani nemusí nikdy doběhnout.

Díky za pozornost!

Budeme rádi za Vaši
zpětnou vazbu! →



[https://goo.gl/forms/
7WHQfs9PJHTWbz2z1](https://goo.gl/forms/7WHQfs9PJHTWbz2z1)