# Podklady

**Já:**Další dotaz směřuje spíš k osnově toho, \*co a jak\* začít psát, případně jak to řadit, tady je nástřel témat a k nim první myšlenka co mi napadla.

1. Definice problému – nechceme mít \_xy\_ instancí, kde je jeden kód + každej server by musel být výkonej (protože plánování), což ne vždycky je možné

2. Proč ten problém vůbec je problém – pálíme zbytečné zdroje, které jsou využívány jen při plánování (odhad 1 % času? - můžu změřit na CAPA), ale pak při samotném běhu aplikace ne

3. Pár nápadů na řešení, kde bych vypíchnul, že nejlepším přístupem je výpočet delegovat pryč na jedno centrální místo

4. Návrhy technických řešení – DSL/Remote Execution/Microservices/atd. určitě to půjde hodně způsobama

**Ondra:**

Ad review mod - myslim, ze je rozumny navrh 1+2. Nasdilej mi repo s kodem, kde uvidim zmeny a budu ti moct delat code review. Na druhou stranu mi prosim pro nejaka vetsi review vygeneruj PDF a posli mailem, protoze si to muzu vytisknout a cist text plynule bez LateXoveho bordelu.

Co se tyce postupu, tak urcite nadefinovat problem (tj. 1 a 2, kde 2 je motivacni kapitola), dale bych si udelal reserzi, jak se to resi, co jsou pribuzne problemy a technologie. Potom by ti z toho mely vypadnout technicke pristupy (4) a ty zacit implementovat.

Typicky pro diplomovy projekt vyzaduji na konci semestru report s 1,2,4 a cast implementace, ktera validuje pristup.

**Já:**

mám celý ten problém (dejme tomu, že to nazveme Remote scheduling) šít na míru TASPu, nebo to vzít obecně a o TASPu se začít zmiňovat až v implementaci?

**Ondra:**  
určitě to definuj obecně, nezávisle na taspu, ale můžeš se inspirovat (parallelizace heuristik). Implementace může pak být proti taspu. Tam bych vyšel z DP Petra Eichlera.

# Definice problému

## Co chci dělat

* Mám generický algoritmus, který nemusí nutné znát doménu nad kterou operuje
  + Opravdu stačí jen interface?
* Mám různé domény a různé problémy
* Chci mít jednu instanci algoritmu, která dokáže pracovat se všemi doménami najednou
* Chci, aby algoritmus byl dobře škálovatelný a co nejvíce obecný
* Chci, aby při přidání nové domény bylo potřeba změnit co nejméně věcí
* Chci ze svého nevýkonného stroje, který neumí plánovat poslat DTO dat, ze kterých následně centrální plánovací server vytvoří plán.
  + Nechci se tedy já jako klient starat o to, kolik výkonu potřebuji na plánování a kolik na samotný chod serveru – obsluhu uživatelů atd.

## Ideální stav

* Server obsluhující uživatele neobsahuje žádný kód plánovače
  + Jeho součástí je jen converter, který databázová data převede na DTO vrstvu, která je následně poslána na plánovací server, který DTO vrstvu převede na datový model se kterým pracuje plánovač
* fire-and-forget přístup – pošlu data v DTO a dostanu zpět plán
* Plánovací server se při přidání nové domény (nová klientské aplikace) modifikuje co nejméně
  + Samotný algoritmus se **nesmí** měnit
    - Scheduler zůstává vždy stejný
  + Je zajištěno, že se aplikace navzájem neovlivňují
    - Instance jsou na sobě **kompletně nezávislé**
* Plánovací server dokáže balancovat svoji zátěž
  + Tedy umí pracovat na několika instancích, kdy existuje centrální server a může existovat i několik dalších instancí, které mohou přebírat buď části, nebo celé úlohy k řešení
* Plánovací server podporuje různé konfigurace plánování
  + Obzvláště výkonnostní
  + Problémy mohou být různě náročné na své výpočetní fáze
    - Inserter/Remover/Evaluator/Scheduler

### TASP related

* Defaultní inserter, Remover
  + Při implementaci aplikace pouze vytvořím converter mezi databázovými daty a TASP modelem
  + Zavolat „plánuj“ a data se odešlou na plánovací server a ten následně vrátí validní plán

## Popis TASPu

* Flow: -> Plan -> Remover -> Inserter -> Evaluator -> Scheduler -> Plan
* Inserter
  + tvoření assignmentů
  + podle benchmarku CAPY zabere nejvíce času právě vkládání assignmentů
    - to je dáno velikostí problému, jiné problémy/domény mohou být rozdílně náročné a potřeby na inserter můžou být odlišné
      * je potřeba aby tohle výsledné řešení umělo rozlišit
  + Na vstupu dostává aktuální plán + evaluátor (je nutné ho tam mít?)
  + Můžou se použít již předpřipravené implementace, nebo implementovat vlastní
    - Jak moc dobře se dají použít generické implementace?
    - Je to skutečně celé generické? -> můžu tam hodit prostě jen „object“?
* Remover
  + Podle benchmarku CAPY nenáročný na výkon, nicméně zase, algoritmus by měl být schopen rozhodnout, kdy bude potřeba výkon i pro Remover a kde ne
  + Stejný interface jako pro Inserter
* Evaluator
* Scheduler
* Plán
  + Je interaktivní, nejen data (je nutné to takhle mít? Nedal by se plán reprezentovat jen jako data?)
  + Immutable – při modifikaci se vytváří vždy nová instance
  + Jak snadno se bude serializovat?

# Motivace proč ho řešit

* Nechci, aby všechny aplikace obsahovaly stejný kód
  + Nepotřebuji celý plánovací algoritmus ve svém kódu, když mi stačí vrstva pro konverzi dat
  + Nechci znovu nasazovat celý server, když potřebuji udělat malou změnu ve způsobu, jakým se plánuje
    - Při nasazení kompletně odstřihnu uživatele, takhle jim „jen“ odstřihnu plánovací funkcionalitu
  + Pokud je plánovací algoritmus IP, které se jen licencuje a neprodává, tak je bezpečnější ho mít mimo zdrojové kódy aplikací, které se prodávají
  + Zjednodušení implementace klientské aplikace
    - Plánovací a klientské části jsou striktně oddělené
      * To umožňuje větší flexibilitu při vývoji aplikací
      * Například plánovač na JVM a server na .NET
    - Klientovi se plánovač jeví jen jako API
* Nechci, aby výkon při plánování snížilo response time serveru, který obsluhuje uživatele
  + Plánování potřebuje hodně výkonu
    - Ten je potřeba právě jen při plánování
    - Když neběží plánování, výkon je nevyužitý
* Při odsunutí plánování jinam nepotřebuji výkonný server pro každou aplikaci, stačí mi jeden, který umí plánovat
  + Není vždy možné mít pro každou aplikaci tak výkonný server, aby zvládl vytvářet „slušné plány“
  + Nechci výkonem plýtvat
    - Když má jedna aplikace výkonný server, ale neplánuje, a druhá, která ho nemá a plánuje, tak velké množství výkonu zůstává nevyužité
* Chci větší modularitu svého modelu – když se rozhodnu udělat změnu v plánovači, nechci shodit celý server obsluhující uživatele
* Chci mít jedno místo, kde běží plánovací algoritmus
  + Díky tomu mám kontrolu nad celkovou alokací zdrojů při plánování
  + Různé aplikace mohou mít různé priority
* Chci mít možnost rozdělit výkon podle fáze algoritmu
  + Například během plánování měnit přidělených zdrojů
  + Inserter dostane více zdrojů než

# Návrhy řešení

Počítám s tím, že kompletně oddělím server, který poskytuje data a server, který data plánuje. Komunikace mezi nimi by měla probíhat po síti. Ať už pomocí RME, nebo klasického REST protokolu, přes který budou proudit serializovaná data

# Poznámky k řešení

* Pokud se bude serializovat plán a přenášet dál po síti, nejlepší bude serializovat jen data/views v konstruktoru **ScheduleManagerImpl** a **UnassignedManagerImpl**
  + Může být problém se serializací lambd