The High Performance Scheduler Game: A Characterization of Slurm, Metrics, and the Viability of Cooperation

Wilmer V. Uruchi Ticona

Universitat Politècnica de Catalunya

2020

Table of Contents

Motivation

- **Slurm** is one of the most widely used **management platforms** for *High Performance Computing* clusters around the world.
- The *users* of these **HPC** platforms are usually involved in **complex projects** of key importance for different areas of **human development**.
- By studying it, we hope to indirectly help those **groups of scientists** that make use of the **HPC** platforms managed by *Slurm*.

Objectives

- The **main objective** of this study is to analyze whether **Algorithmic Game Theory** and specifically **Mechanism Design** tools can be used to build a **model** that **captures** key characteristics of the *Slurm* Scheduler.
- Produce a model that implements some **desired guarantees**, an **ideal** model, so we can achieve optimal results that can be **compared** to **variations** of the model, and the results of a proper **Slurm Simulator**[1].
- We try to get a glimpse at the **Price of Anarchy** of the *Slurm* Scheduler, which is the ratio between the **worst** *possible* outcome and the **best** *possible* outcome. We argue about the **Viability of Cooperation**.

[1] Ana Jokanovic, Marco D'Amico, Julita Corbalan. Evaluating SLURM Simulator with Real-Machine SLURM and Vice Versa. *Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS18)*

Secondary Objectives

- Help *Slurm* developers to get more **ideas** into which way to take the **improvement** of their platform.
- Provide to the **users** of the HPC platforms that implement *Slurm* a **description** of what is happening **behind the scheduling process**, so they can implement **strategies** of their own.
- Present an **intuitive** explanation of **Mechanism Design** concepts by testing them in an **experimental** setting.
- Perform an in-depth analysis of the data generated by **Autosubmit**, a **workflow manager** used by the *Earth Science Department* at *BSC-CNS*.
- Implement the models and provide the source code to reproduce the results.
- Provide a description of the setup process of the **Slurm Simulator**.

Table of Contents

SLURM

- *Slurm* is an open source, fault-tolerant, and highly scalable **cluster management system** for large and small Linux clusters.
- In *MareNostrum4*, there are around 3456 nodes, each node has 2 sockets, each socket has 24 cores or *CPUs*.
- The *Slurm* scheduling mechanism has two main components: **Priority** and **Scheduler**.

Priority

A value calculated based on **data** from the **user**, and the **jobs** that the user sends to the High Performance Computer (HPC). The calculation tries to give **higher priority** to those users that have **less usage**.

1. **Age**: The longer a job sits in the queue and is eligible to run, the bigger this value gets.
2. **Size**: This value is determined by the number of processors (*CPUs*) a job requests. We consider **nodes**.
3. **Fair-share**: Determined by an algorithm that takes as input the number of users and the representation of the user hierarchy. Considers *Usage*.
4. **Quality of Service** *QoS*: Represents a set of rules that apply to the jobs sent using it.

Scheduler

- The **Scheduler** runs in interval of 60 seconds.
- Considers the order imposed by the **Priority** value of each job.
- Favors **resource and time optimization** to decide which job is executed next.
- Uses **backfill mode**, where the scheduler will start lower priority jobs if that does not delay the start of higher priority jobs.

Backfill Scheduling

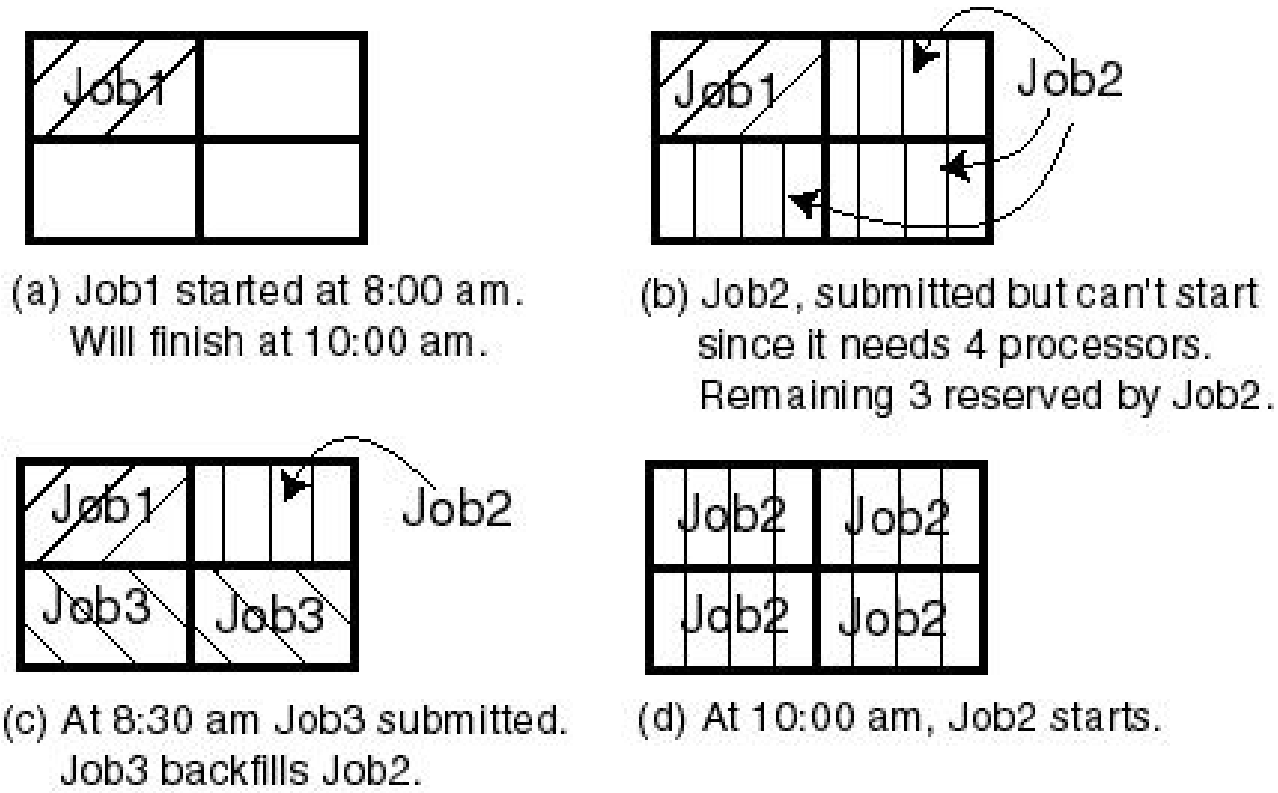

Figure: **Backfill** Scheduling. Image taken from *IBM Knowledge Center*.

Autosubmit

Slurm receives **jobs** that come from **experiments**. A typical experiment can be seen as a **Directed Acyclic Graph**, it starts with jobs that retrieve or send information, followed by more complex jobs.

- A workflow manager for experiments.
- Implemented as a *Python* library that handles authentication, job submit, retrieval of job results, etc.
- Stores the the time a job was **submitted, started, finished**, and the **resources** requested.

Data Analysis Summary

We start by obtaining and analyzing the data at an experiment level. After identifying the useful experiments, we proceed to analyze the data at a **job level**. Then, we clean the job data to finally obtain our datasource.

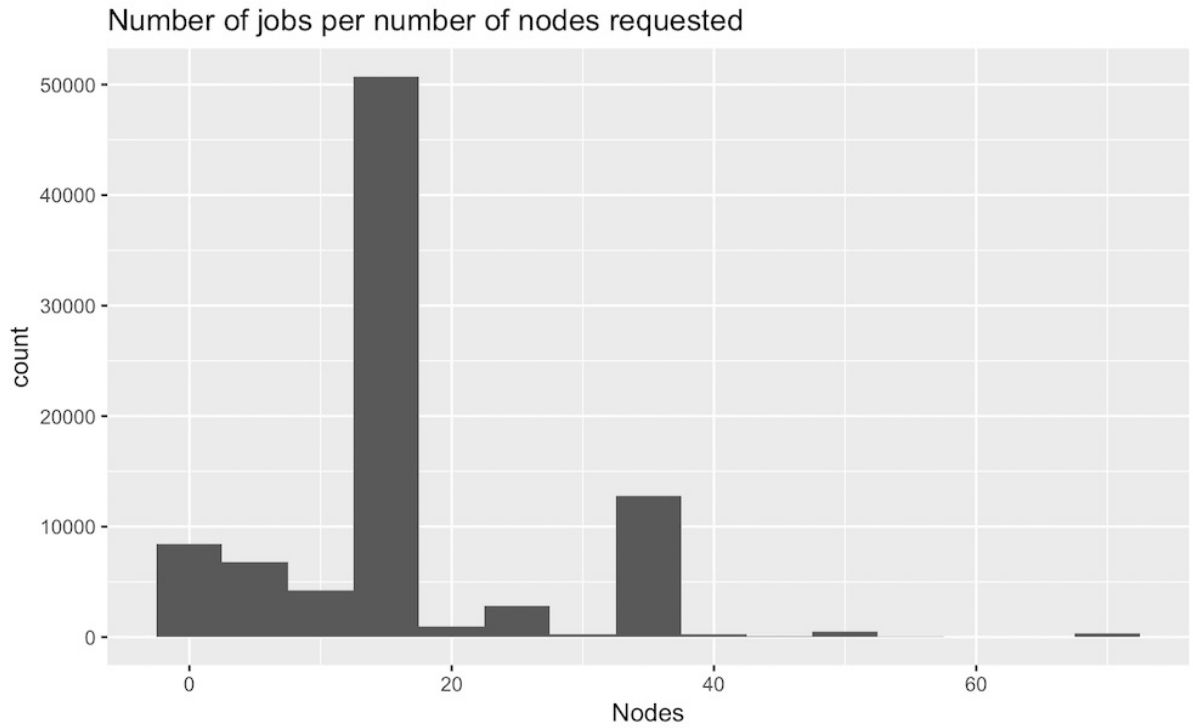

Figure: Number of jobs per nodes requested. *binwidth* = 5.

Game Theory

The **Algorithmic Game Theory** approach gives us an **economic** perspective that we can use to understand the effect of **user interaction** and **selfish behavior** into a model where **users compete** for **limited resources**.

The inclusion of **Mechanism Design** is key to this objective because this field gives us the **tools** to **design** an **ideal** mechanism that can produce an optimal result and the tools to **analyze** of it.

General Concepts

- **Game**: A set of circumstances that has a **result dependent** on the **actions** of players.
- **Strategy**: *One of the options* which the player can choose. It can be **deterministic (pure)** or **probabilistic (mixed)**.
- **One-shot Simultaneous Move Game**: The individual reward depends on this join selection.
- **Dominant Strategy**: Any other option would just mean, in the best case, **an equal utility**. A game has a **dominant strategy solution** if all players have a **dominant strategy**. *Few games have a dominant strategy solution*.

General Concepts

- **Social Welfare**: The **aggregation** of the players **values**.
- **Mixed Nash Equilibrium**: A **stable solution**, so that any player cannot deviate individually and obtain a higher benefit.
- **The Price of Anarchy**: Defined as the **ratio** between the **worst** value of the **social objective function** of an equilibrium and the **optimal** value of the same social objective function.

Mechanism Design

The science of **rule-making**. This sub-field of economic theory, which has an **engineering perspective**, is interested in designing mechanisms that achieve a **socially desirable outcome**, or a desirable **property defined by the designer**.
We use **Mechanism Design** to produce an **ideal model** against which we can **compare variants** of it and the results from the **Slurm Simulator**.

Mechanism Design

Auction: A mechanism that **allocates** items and defines **payments**. These decisions are formalized as an **allocation rule** to define **who receives what**, and a **payment rule** that defines **how much the winning players have to pay** to the mechanism.
Vickrey Auction: *Second Price Auction*, the **single item is awarded to the highest bidder**, and **pays** an amount equal to the **second highest bid**. The **dominant strategy** is to **bid truthfully (bid = value)**, even without knowing what other players bid.
Direct-revelation mechanism: players reveal their private information to the mechanism. **Dominant strategy** is to bid truthfully.

Mechanism Design

We are looking for an ideal auction (model) that has these properties:

- **Strong incentive guarantees:** **Truthful bidding** is a **dominant strategy** and **never leads to negative utility**. We refer to mechanisms that comply with this guarantee as *Dominant Strategy Incentive Compatible* (**DSIC**).
- **Strong performance guarantees:** **Social Welfare** maximization, the bidder with the highest value gets the item, assuming truthful bids.
- **Computationally efficient:** The **allocation** and **payments** can be computed in polynomial time (or even linear) as a **function of the bids**.

Knapsack Auction[3]

Each task has a publicly known size w_i and a privately known valuation. The system has capacity W. The **feasible set** X is the 0-1 n-vector $(x_1, x_2, ..., x_n)$ such that, $\sum_i w_i x_i \leq W$, where $x_i = 1$ if i is a winning bidder. Allocation rule:

$$\mathbf{x(b)} = argmax_X \sum_{i=1}^{n} b_i x_i$$

Define a **payment rule** that extends the **allocation rule** to *DSIC*: Players **pay** the **marginal harm** their bid causes; the payment can be **as high as their original bid (VCG payments)**[2].
Not computationally efficient in general, but we can use an **approximation**.

[2] Generalization of Vickrey Auction.

[3] Mu'Alem, A. and Nisan, N. (2008). Truthful approximation algorithms for restricted combinatorial auctions. *Games and Economic Behavior*, 64(2):612-631.

Greedy Knapsack Algorithm

Consider a bid profile **b** and a set of winners I with total size $\sum_{i \in I} w_i \leq W$. For all $i \in I$, we have that $w_i \leq W$. Then we follow the steps:

1. Sort and re-index the bidders so that:

$$\frac{b_1}{w_1} \geq \frac{b_2}{w_2} \geq \cdots \geq \frac{b_n}{w_n}$$

2. Pick items in that order until **one does not fit and halt**.
3. Return the solution from the previous step or the highest bidder, whichever has largest social welfare.

Social Welfare achieved by the greedy algorithm is **at least** 50% of the maximum **Social Welfare**. **Better** results if $w_i \leq \beta \cdot W$ for every player i and $\beta \in (0, \frac{1}{2}]$. Then, the approximation guarantee increases to $1 - \beta$.

Table of Contents

Knapsack Auction Model

- Measure performance as the **maximization of an objective function**: **Total Value** (**Social Welfare**) achieved by the scheduler, which is the sum of the values each user has for the execution of her job.
- **Player** i submits job i that has a known size w_i representing the **number of nodes** required, and t_i representing the **minutes** required to finish execution (**planned running time**).
- Considering w_i and t_i, player i has a **privately known value** v_i. The values v_i of each player are represented by their **reported bids** $b_1, b_2, ..., b_n$. The **dominant strategy** is to bid **truthfully**, $b_i = v_i$.
- The **allocation rule** maximizes **Social Welfare**, and it is **computationally efficient** by using the **Knapsack Greedy Algorithm**.
- The **payment rule** uses **VCG payments** to ensure a *DSIC* mechanism.

Knapsack Auction Modified

For **Knapsack Greedy Algorithm**, keep going until a job that **fits the remaining capacity** is found or end of list. This breaks **DSIC**.
Is **Social Welfare** no longer maximized?

Knapsack + Priority

If we base our **allocation rule** in a randomly generated **Priority** value, then, our **payment rule does not make sense anymore**.
What happens to **Social Welfare**?

Experimentation

For experimentation purposes, we will choose jobs uniformly at **random** from the datasource until they sum no less than **6912 nodes**.

1. W: Total n**umber of nodes** in the HPC (3456).
2. n : **Number of players** (jobs).
3. w_i : **Weight** of job i measured in nodes.
4. b_i : **Bid (Value)** of player i for job i.
5. p_i : **Payment** of player i for job i.
6. x_i : 1 if job i is **selected** into the knapsack, 0 otherwise.
7. min_i : **Planned time** in minutes that player i considers job i will take in the worst case.
8. $rtime_i$: **Time in minutes** that player i considers job i will take. Previous experience running similar jobs gives player i a better idea of how much time the job will really take, so $min_i \geq rtime_i$.

Results metrics

1. **Total Value**: *The sum of the bids* of all selected players, also known as the **Social Welfare**, $\mathbf{TV} = \sum_i^n b_i \cdot x_i$.
2. **Total Sum of Payments**: *The sum of the calculated payments* for all selected players, $\mathbf{TP} = \sum_i^n p_i \cdot x_i$.
3. **Pay More Count**: *Number of players* that would pay more than their bid, $\mathbf{PM} = |\{i : x_i \geq 1 \wedge p_i > b_i\}|$
4. **Same Payment Count**: *Number of players that would pay the same amount as their bid*, $\mathbf{SP} = |\{i : x_i \geq 1 \wedge p_i = b_i\}|$

Experimentation Space

In **Experiment 1A** we take $b_i = w_i \cdot min_i$. Then, we evaluate the results using a solver that finds the optimal **Total Value**.
In **Experiment 1B** we follow the same formula for b_i used in **Experiment 1A**; however, instead of using an optimal solver, we use the *Knapsack Greedy Algorithm*.
In **Experiment 3B** we take $b_i = w_i \cdot min_i$ to represent an *inexperienced* player. Then, the greedy algorithm uses $\frac{b_i}{w_i \cdot rtime_i}$.
In **Experiment 2B** we take $b_i = (w_i \cdot rtime_i) + (w_i \cdot rtime_i) \cdot 0.1$. Then, the greedy algorithm uses $\frac{b_i}{w_i \cdot rtime_i}$.
In **Experiment 4B**, we take $b_i = (w_i \cdot rtime_i) + (w_i \cdot rtime_i) \cdot 0.1$. Then, the greedy algorithm uses $\frac{b_i}{w_i}$, as originally defined.

Results: Main Experimentation

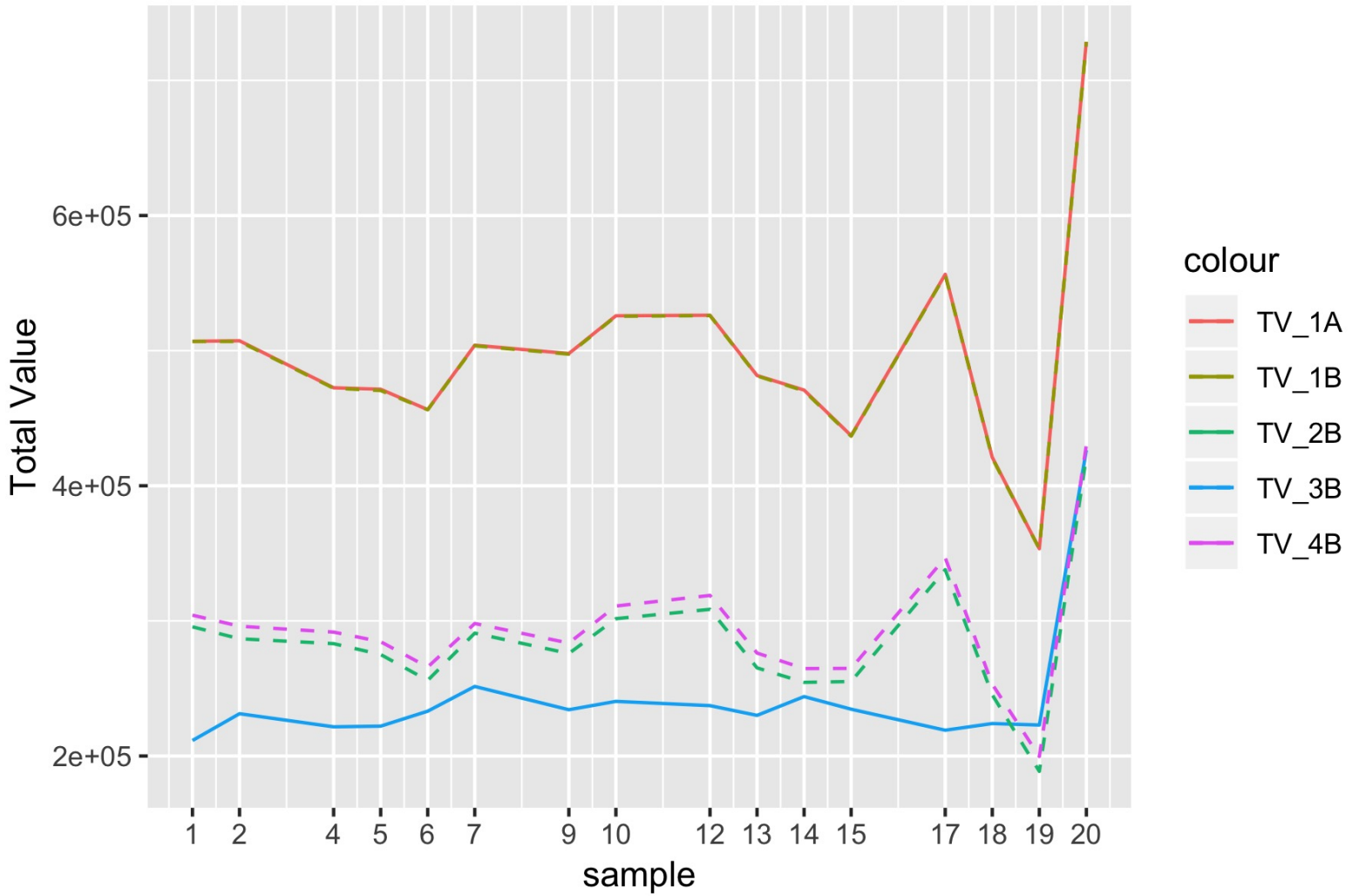

Figure: Total Value (Social Welfare) for sample and experiment.

Results: Main Experimentation

	sample	TV_1A	TV_1B	TV_3B	TV_2B	TV_4B
1	sample_1.txt	**506930**	506850	211480	295615	**304239**
2	sample_2.txt	**507305**	506785	231310	286779	**296064**
3	sample_4.txt	**472530**	472170	221610	283158	**291715**
4	sample_5.txt	**471440**	470480	222105	275120	**284591**
5	sample_6.txt	**456315**	456275	233180	255977	**265832**
6	sample_7.txt	**504035**	503595	251495	290940	**298085**
7	sample_9.txt	**497850**	497410	234255	275866	**283595**
8	sample_10.txt	**525810**	525450	240445	301627	**310999**
9	sample_12.txt	**526210**	525930	237285	308551	**318843**
10	sample_13.txt	**481620**	481300	230075	265248	**276205**
16	sample_20.txt	**728540**	728540	426377	421757	**430275**

Table: Total Value per sample and experiment.

Results: Main Experimentation

	sample	TP_1A	TP_1B	TP_3B	TP_2B	TP_4B
1	sample_1.txt	**138240**	122320	118400	62792	**64796**
2	sample_2.txt	**138240**	118400	119700	60788	**63460**
3	sample_4.txt	**138240**	119680	115840	**63794**	**63794**
4	sample_5.txt	**138240**	12640	114680	60120	**62792**
5	sample_6.txt	**138240**	113280	113920	60454	**62124**
6	sample_7.txt	**138240**	118400	113380	61122	**63126**
7	sample_9.txt	138240	117760	**456960**	61456	**62792**
8	sample_10.txt	**138240**	117760	117120	59786	**62458**
9	sample_12.txt	**138240**	112000	106880	58784	**60454**
10	sample_13.txt	**138240**	117760	113920	60454	**63126**
16	sample_20.txt	**207360**	171840	8778	2465	**30496**

Table: Total Payments per sample and experiment.

Results: Main Experimentation

	sample	PM_1A	PM_1B	PM_3B	PM_2B	PM_4B
1	sample_1.txt	0	0	4	5	0
2	sample_2.txt	0	0	6	10	0
3	sample_4.txt	0	0	4	8	0
4	sample_5.txt	0	0	5	9	0
5	sample_6.txt	0	0	7	10	0
6	sample_7.txt	0	0	5	5	0
7	sample_9.txt	0	0	194	6	0
8	sample_10.txt	0	0	2	10	0
9	sample_12.txt	0	0	2	6	0
10	sample_13.txt	0	0	5	8	0
16	sample_20.txt	0	0	0	0	0

Table: Number of players that will pay more than their bid per sample and experiment.

Results: Main Experimentation

	sample	SP_1A	SP_1B	SP_3B	TV_2B	SP_4B
1	sample_1.txt	82	74	121	75	**92**
2	sample_2.txt	75	65	111	64	**85**
3	sample_4.txt	61	54	117	62	**70**
4	sample_5.txt	69	1	118	52	**74**
5	sample_6.txt	58	53	101	48	**68**
6	sample_7.txt	68	56	109	68	**79**
7	sample_9.txt	**72**	65	1	56	67
8	sample_10.txt	**65**	58	122	41	61
9	sample_12.txt	63	53	108	56	**71**
10	sample_13.txt	59	50	105	53	**66**
16	sample_20.txt	**22**	13	22	1	2

Table: Number of players that will pay their bid per sample and experiment. We can ignore the results for experiment **3B** and **2B**.

Results: Knapsack Auction Modified

Instead of stopping when a job that does not fit is found, the greedy algorithm continues until it **fills the capacity** of the Knapsack or reaches the end of the list.
We define **1B1** and **4B1** as the experiments **1B** and **4B** from **Main Experimentation**, respectively.

Results: Knapsack Auction Modified

	sample	TV_1B	TV_1B1	TV_4B	TV_4B1
1	sample_1.txt	**506910**	506850	**304308**	304239
2	sample_2.txt	**507265**	506785	**296247**	296064
3	sample_4.txt	**472530**	472170	**291784**	291715
4	sample_5.txt	**471415**	470480	**284609**	284591
5	sample_6.txt	**456275**	**456275**	**265996**	265832
6	sample_7.txt	**504015**	503595	**298241**	298085
7	sample_9.txt	**497830**	497410	**283663**	283595
8	sample_10.txt	**525750**	525450	**311160**	310999
9	sample_12.txt	**526125**	525930	**318944**	318843
10	sample_13.txt	**481600**	481300	**276205**	**276205**
16	sample_20.txt	**728540**	**728540**	**430354**	430275

Table: Total Value compared between the Knapsack Greedy Algorithm Modified **TV_1B TV_4B** and **TV_1B1 TV_4B1** from the Knapsack Greedy Algorithm.

Results: Knapsack Auction Modified

	sample	TP_1B	TP_1B1	TP_4B	TP_4B1
1	sample_1.txt	**138080**	122320	**71970**	64796
2	sample_2.txt	**137240**	118400	**76399**	63460
3	sample_4.txt	**135300**	119680	**72033**	63794
4	sample_5.txt	**139795**	12640	**68488**	62792
5	sample_6.txt	**134340**	113280	**76356**	62124
6	sample_7.txt	**138905**	118400	**73547**	63126
7	sample_9.txt	**137675**	117760	**70561**	62792
8	sample_10.txt	**140525**	117760	**77461**	62458
9	sample_12.txt	**142705**	112000	**71760**	60454
10	sample_13.txt	**138730**	117760	**70106**	63126
16	sample_20.txt	**205935**	171840	**37386**	30496

Table: Total Payment compared between the modified version **TP_1B TP_4B** and the original version **TP_1B1 TP_4B1**

Results: Knapsack Auction Modified

	sample	PM_1B	PM_1B1	PM_4B	PM_4B1
1	sample_1.txt	2	0	0	0
2	sample_2.txt	3	0	5	0
3	sample_4.txt	0	0	0	0
4	sample_5.txt	**71**	0	0	0
5	sample_6.txt	0	0	8	0
6	sample_7.txt	4	0	2	0
7	sample_9.txt	2	0	0	0
8	sample_10.txt	6	0	2	0
9	sample_12.txt	10	0	0	0
10	sample_13.txt	0	0	0	0
16	sample_20.txt	0	0	0	0

Table: *Number of players that will pay more than their bid* compared between the modified version **PM_1B PM_4B** and the original version **PM_1B1 PM_4B1**

Results: Knapsack + Priority

Use **Priority** to decide which jobs are considered for **resource allocation**. This **Priority** is decided based on the formula: $priority_i = \frac{w_i \cdot 10000}{W} + y$, where y is a uniform random variable between 1 and 10000. We apply this formula to **mimic in some way** what the *Slurm* calculation of **Priority** would do. We compare the **Total Value** of the outcome from experiments **1B** and **4B** with previous results.
We define **1B1** and **4B1** as the experiments **1B** and **4B** respectively from **Main Experimentation**.

Results: Knapsack + Priority

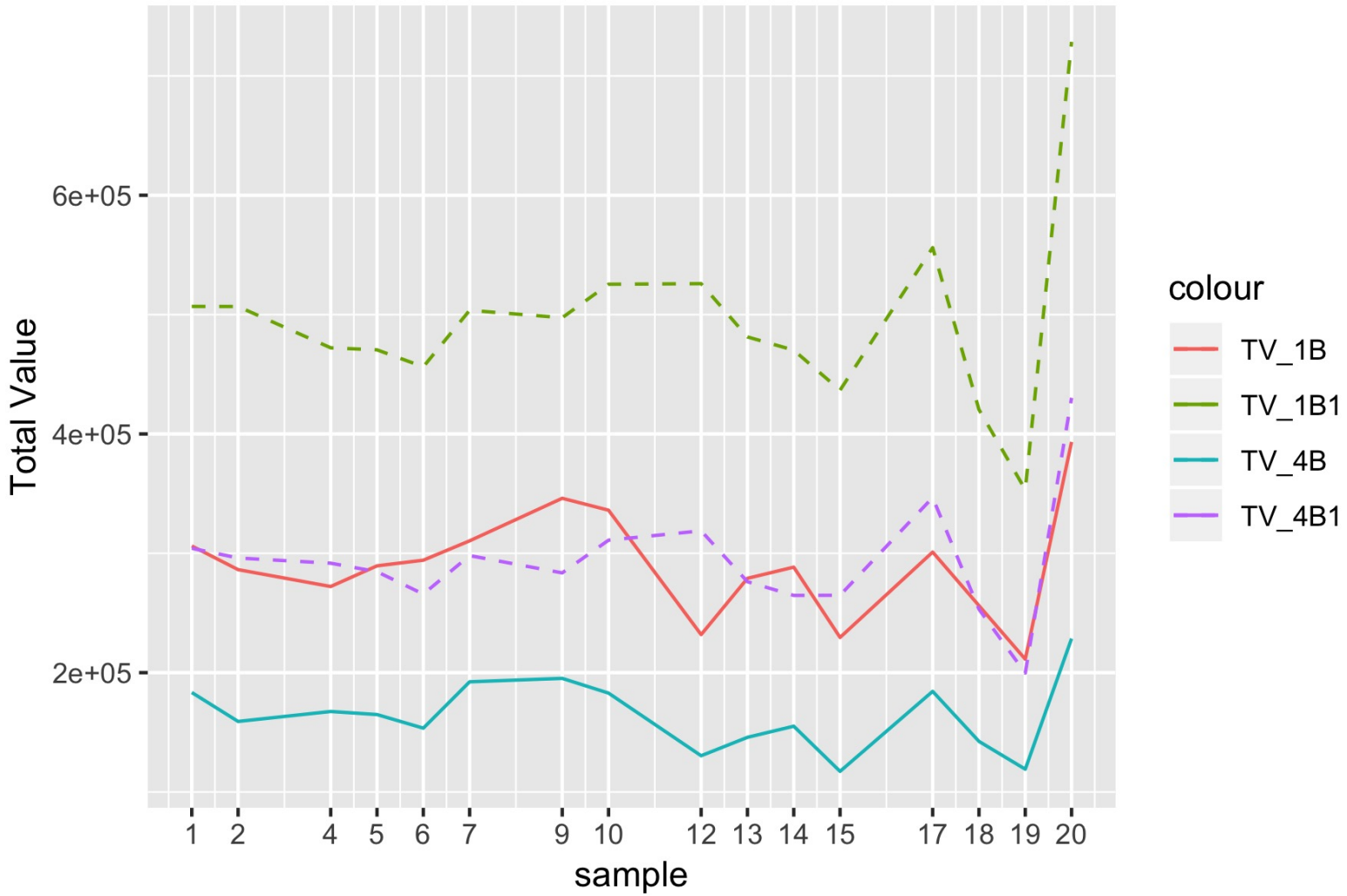

Figure: Total Value per sample and experiment using Priority.

Results: Slurm Simulator

- Use the **Slurm Simulator** to run our samples and produce what would be a real output from the *Slurm* Scheduler.
- Use the formula $v_i = w_i \cdot min_i$ to calculate the value for each job i and get a **Total Value** achieved by the resource allocation outcome of the simulator.
- Then, we compare it to the experiments **1B** from **Main Experimentation** that we call **1B1**, **1B** from **Knapsack Auction Modified** that we call **1B2**, and **1B** from **Knapsack + Priority** that we call **1B3**.

Results: Slurm Simulator

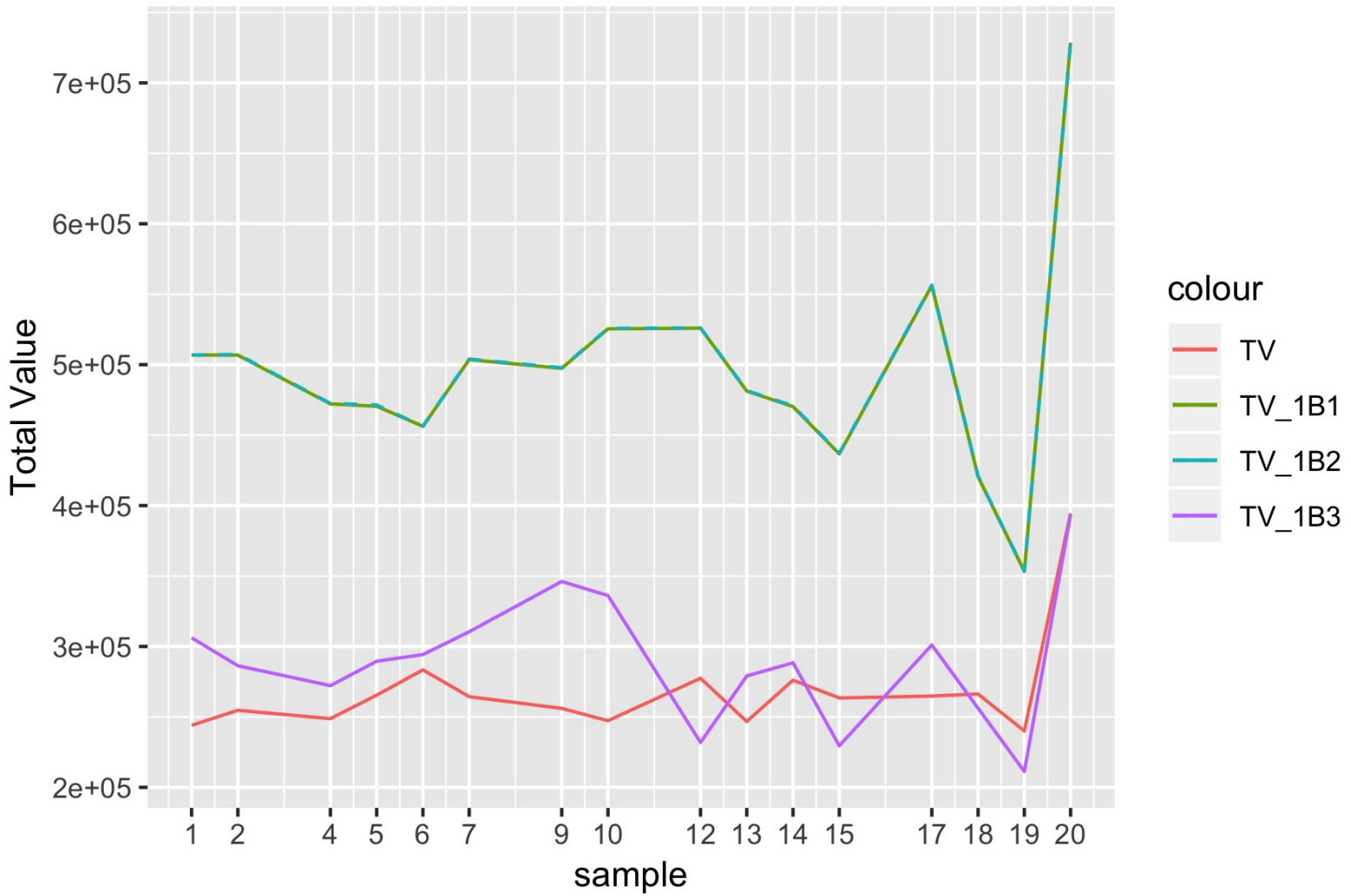

Figure: Total Value per sample and experiment from the *Slurm* Simulator compared to previous results.

Price of Anarchy

	sample	TV	TV_1B	PoA
1	sample_1.txt	244007	506850	2.08
2	sample_2.txt	254720	506785	1.99
3	sample_4.txt	248748	472170	1.90
4	sample_5.txt	265446	470480	1.77
5	sample_6.txt	283402	456275	1.61
6	sample_7.txt	264376	503595	1.90
7	sample_9.txt	256051	497410	1.94
8	sample_10.txt	247317	525450	2.12
9	sample_12.txt	277427	525930	1.90
10	sample_13.txt	246755	481300	1.95
16	sample_20.txt	394505	728540	1.85

Table: Price of Anarchy based on Total Value of our samples.

Price of Anarchy

- Through experimentation we have seen that it is possible to achieve a **near optimal Total Value** for our model in a **computationally efficient way**.
- We calculated the **Total Value** for the **results of the simulator** based on the *same ideas applied for our near optimal model*.
- We have established a **common ground**, the **Total Value**, that we can use to compare the outcome of our **ideal model** to the outcome of the **simulator**.
- If we average the **PoA** results, we get that $PoA \approx 1.85$. Then, we can say that an ideal outcome of the **Slurm scheduling game** might be around 1.8 **times better** than a **usual** outcome.

Viability of Cooperation

What if the dominant strategy was to inevitably **cooperate** *to get a* **greater utility** *or a* **lower cost***?*

- The **Knapsack Auction Model** is an ideal game or auction.
- The **dominant strategy** for the players is to bid **truthfully**.
- The mechanism achieves **Social Welfare** maximization.
- **Telling the truth**, or being forced to tell the truth, is also a form of **cooperation**.
- The **Knapsack Auction Model** imposes the necessary constraints on users to achieve this form of **cooperation**.
- It is not trivial how to translate the desired guarantees achieved by the **Knapsack Auction Model** to the **Slurm Scheduler**. The **planned** running time that the player establishes for her job plays a **key role** in the **scheduling** operation.

Table of Contents

Conclusions

- We developed and presented a **detailed study** of the **main factors** that play in the **scheduling process of Slurm**.
- We present a **summary** of some of the main ideas we used for our project from the field of **Mechanism Design**, also including relevant **background** theory from **Algorithmic Game Theory**
- The **payment rule** defined by *VCG payments* together with the **allocation rule** make our model *DSIC*.
- We have designed a model that **captures** some of the most important **characteristics** of the *Slurm* Scheduler.

Conclusions

- Our *ideal* auction, the **Knapsack Auction Model**, achieves the **best** results regarding **resource allocation** in **reasonable time**, among other desired guarantees.
- From the **variants** of our model, the **Knapsack Greedy Algorithm Modified** is the one that best resembles the **backfill** procedure that attempts to allocate as many available resources as possible.
- We have seen that an **allocation rule** based on **Priority**, which attempts to allocate resource in a **fair** way, achieves a **lower Social Welfare** than that of a mechanism that implements an **allocation rule** that gives resources to those **who value them the most**.

Future Work

- **Experimentation** has hinted that the **distribution** of the jobs that arrive at the scheduler plays an important role in the outcome of the **allocation rule**. A next step is to pursue more analysis of these distributions to find if some of them **favor certain outcomes**.
- The inclusion of some kind of **currency** in the calculation of **Priority** by the **Slurm Scheduler** might result in **better guarantees** for achieving a greater **Social Welfare**.
- We have looked at results for players that **pay more** than their bid, or that **pay the same**, but we have not analyzed the results for players that **pay less** than their bid.
- Analyze under which **conditions** a **worst equilibrium** might occur.

Thank you.