Russian Supercomputing Days 2025

# Efficient Allocation of Resources under Group Dependencies and Availability Uncertainties

Victor Toporkov

Dmitry Yemelyanov

and Artem Bulkhak

National Research University “Moscow Power Engineering Institute”

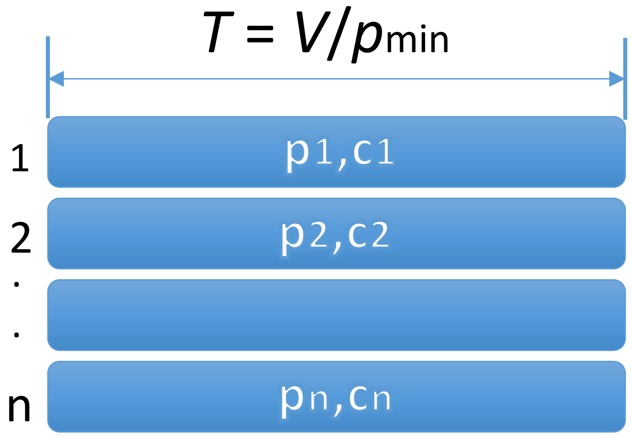
# HPCS Scheduling Problem


- HPCS and Virtual Organizations should implement efficient procedures for job-flow scheduling, execution and the resources allocation

# Job Resource Request and Parallelization

The resource requirements for a single parallel job execution are arranged into a resource request:

- ***n*** - number of simultaneously requested computational nodes
- ***p*** - minimal performance requirement for each computational node
- ***V*** - average computational volume for a single node process
- ***C*** - maximum total job execution cost (budget)

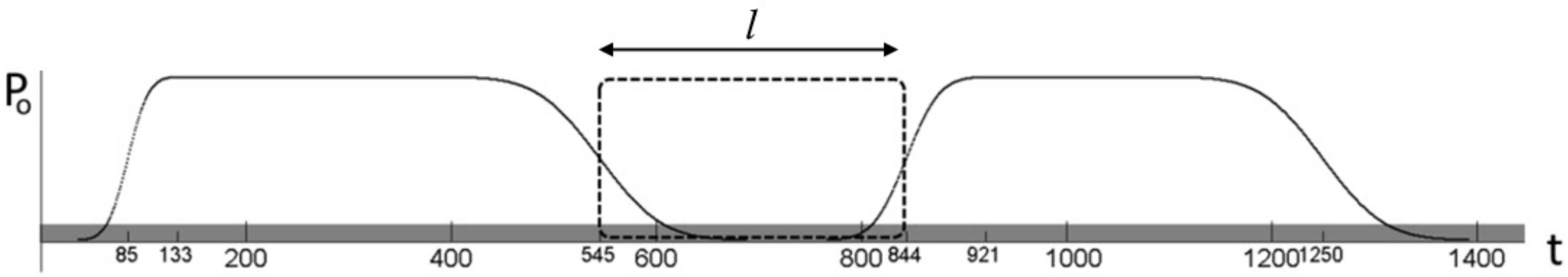



Allocate *n* simultaneously *available* resources for a time interval *T* with a total cost constraint *C*

# Availability-based Scheduling

- Traditional models consider job-flow scheduling problem in a deterministic way
  - About 20% of Grid computational nodes exhibit truly random availability intervals
- In this work the uncertainties are modeled as resources availability events and probabilities: a natural way of machine learning and statistical predictions representation
- Resources availability predictions may originate from:
  - Historical data
  - Linear regression models
  - Expert and machine learning systems
  - Expert and user estimations

# Job Scheduling Under Uncertainties




- Each single node is characterized with a set of availability events
  - Availability event can be described with a random variable distribution
- The node availability probability $P_a$ during the whole interval $l$ depends on the occupation events
- In order to execute a parallel job a set of nodes (a window) should be allocated simultaneously
- We want to maximize a total window availability probability

# Window Availability Calculation

The total window availability probability may be calculated as follows:

$$P_a^w = \prod_i^n P_a^{r_i} \to \max,$$

Allocate a set of $n$ nodes with a total cost constraint:

$$\sum_{i=1}^n c_i \leq C$$

Dynamic Programming solution (0-1 multiplicative knapsack):

$$f_j(c, v) = \max\{f_{j-1}(c, v), f_{j-1}(c - c_j, v - 1) \ * \ P_j\}$$

$$j = 1, .., m, c = 1, .., C, v = 1, .., n,$$

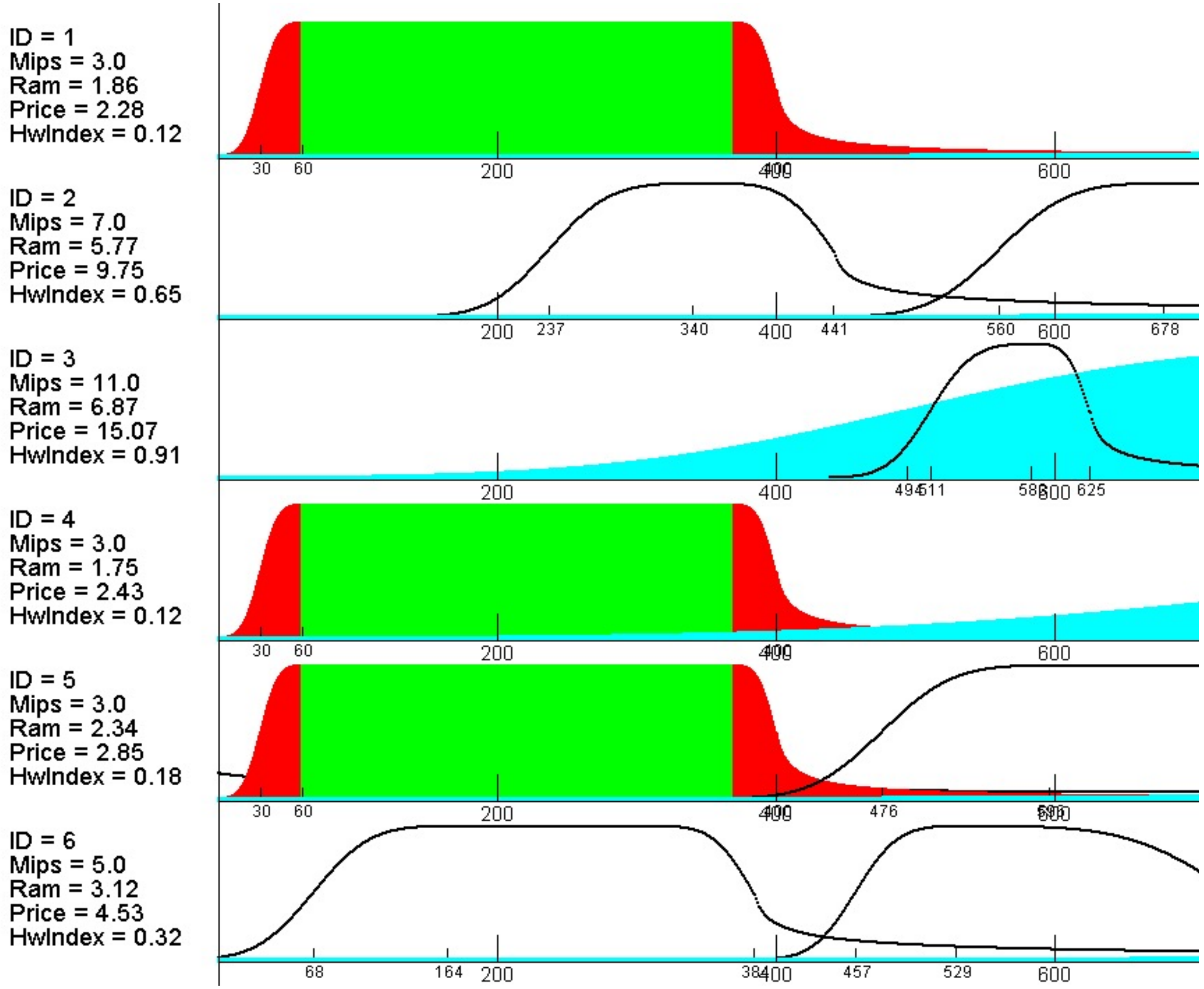
# Group Dependencies between the Resources

In general, the resources and their utilization events *are not independent*

Groups $G_i \in G$ represent subsets of resources sharing common properties

Group examples:

- Resources of a parallel job share common utilization events
- Discount provided for resources selected from the same vendor
- Performance benefits for matching resources
- Geographical location and connectivity-based groups



# Group Dependencies Formalization

- $P_a^{\mathrm{G_i}}$ is a common availability probability for group $G_i$ (during interval $l$)
- If at least one resource from group $G_i$ is selected for window W, then the common probability $P_a^{\mathrm{G_i}}$ is included into total availability $P_a^w$:

$$P_a^w = \prod_i^{n^*} P_a^{G_i}$$

where $n^*$ is a number of diverse groups used for the window $W$

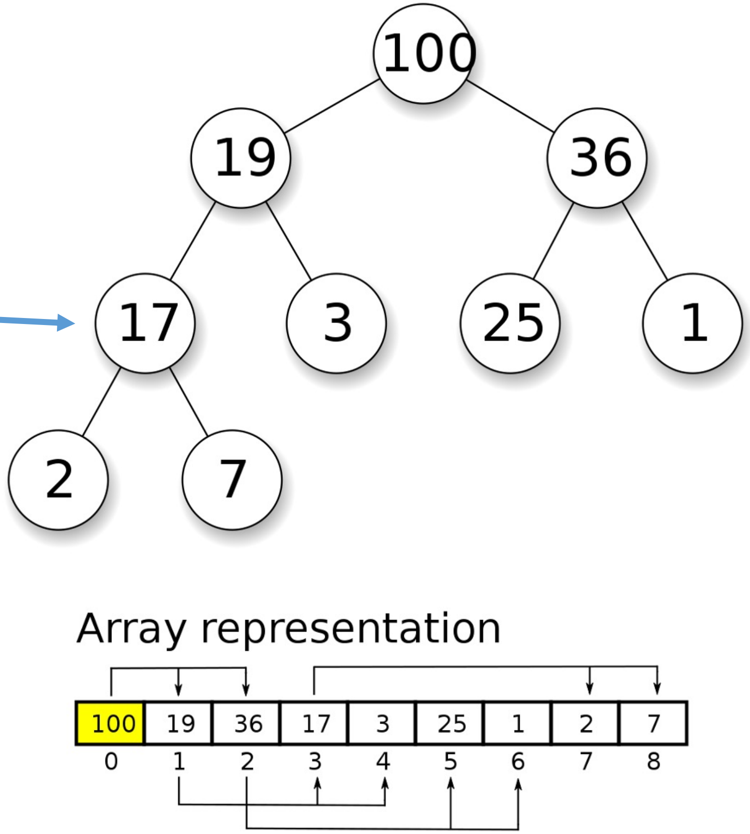
- Total cost constraint: $\sum_{i=1}^{n} c_i \leq C$

# Branch and Bounds Algorithm to Address Group Dependencies

- We implement branch-and-bounds approach to consider resources groupings for the resources' selection

- Max-Heap data structure is maintained for the solution tree

- For each solution node we maintain:
  - $G^+$ set of groups to be included in the solution
  - $G^-$ set of groups to be excluded from the solution
  - Other groups
  - (Upper estimate) criterion value $P_a^w$



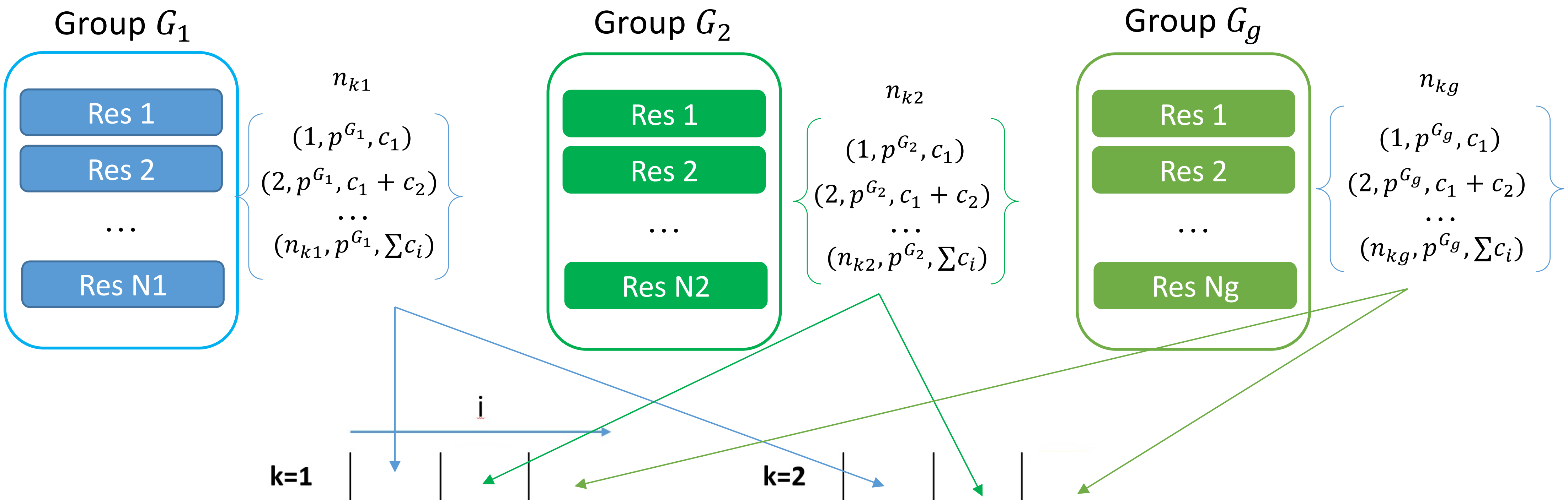
# Group Knapsack Algorithm (GKA)

- GKA considers groups of resources $G_i$ as enumeration items instead of individual VMs
- Instead of a single pair of characteristics $p_i$ and $c_i$, each group item $G_i$ provides a list of $NV_i$ possible resource allocation *variants* $Var_j = (n_j, u_j, c_j)$
- GKA iterates over groups $G_i \in G$ and their variants $\{Var_j\}$ to calculate the following recurrent scheme:

$$f_i(c,n) = \max\{f_{i-1}(c,n), f_{\mathrm{i-1}}(c - c_j, n - \mathrm{n_j}) + u_j\},$$
$$i = 1,..,|G|, j = 1,..,NV_i, c = 1,..,C_{\max}, n = 1,..,n_{\max}$$

- $f_i(c,n)$ then maintains the maximum possible *aggregate* utility $U$ achievable for a subset of $n$ VMs combined from different variants from groups $\{G_1,..,G_i\}$ for a budget $c$
- Estimated computational complexity is bounded by $O(N * \mathrm{n_{max}} * C_{\max})$

# Group Knapsack Algorithm (GKA)




C

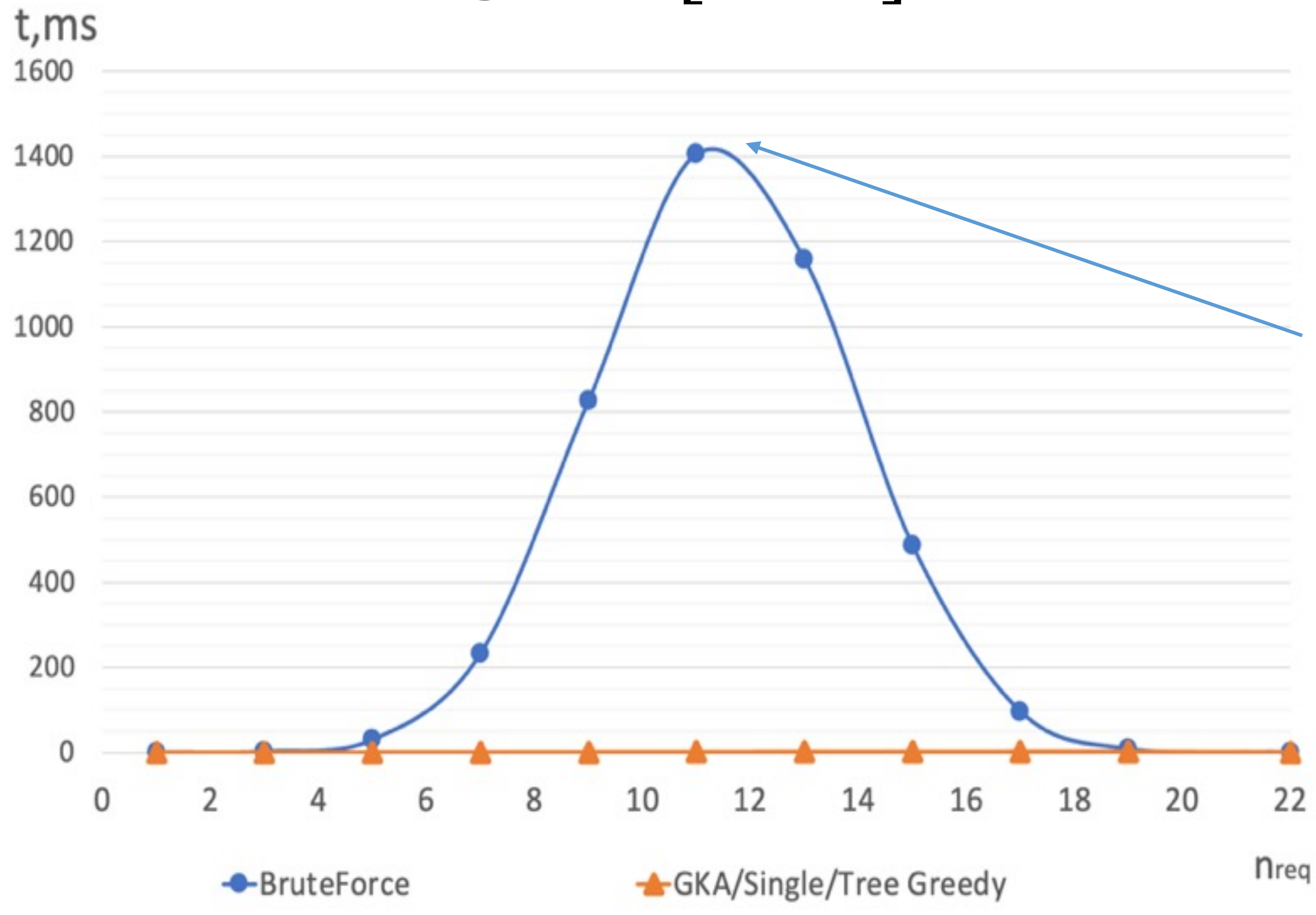
| k=1 | | | |
|---|---|---|---|
| 0 | - | - | - |
| 1 | - | - | - |
| 2 | - | 1 | 1 |
| 3 | 3 | 3 | 3 |
| 4 | 3 | 3 | 6 |
| 5 | 3 | 3 | 6 |
| 6 | 3 | 3 | 6 |

| k=2 | | | |
|---|---|---|---|
| 0 | - | - | - |
| 1 | - | - | - |
| 2 | - | - | - |
| 3 | - | - | - |
| 4 | - | - | - |
| 5 | - | 4 | 4 |
| 6 | - | 4 | 7 |

# Algorithms for Analysis and Comparison

- Brute Force provides exact solution but usually not feasible for $N > 35$ resources
- Knapsack Single implements resources allocation for $P_a^w = \prod_i^n p_i \to \max$ without any knowledge of the resources' groupings (multiplicative knapsack)
- Exact Branch and Bounds (Tree) implements the presented branch-and-bounds approach with *Knapsack Single* for intermediate calculations, thus providing exact solution in integers
- Greedy Branch and Bounds (Tree Greedy) implements the same branch-and-bounds approach but uses more performance-efficient greedy approximation for the intermediate calculations
- Group Knapsack (GKA) implements group 0-1 knapsack with account for group dependencies

# Execution Time
# selecting $n \in [1; 22]$ from $N = 22$ resources



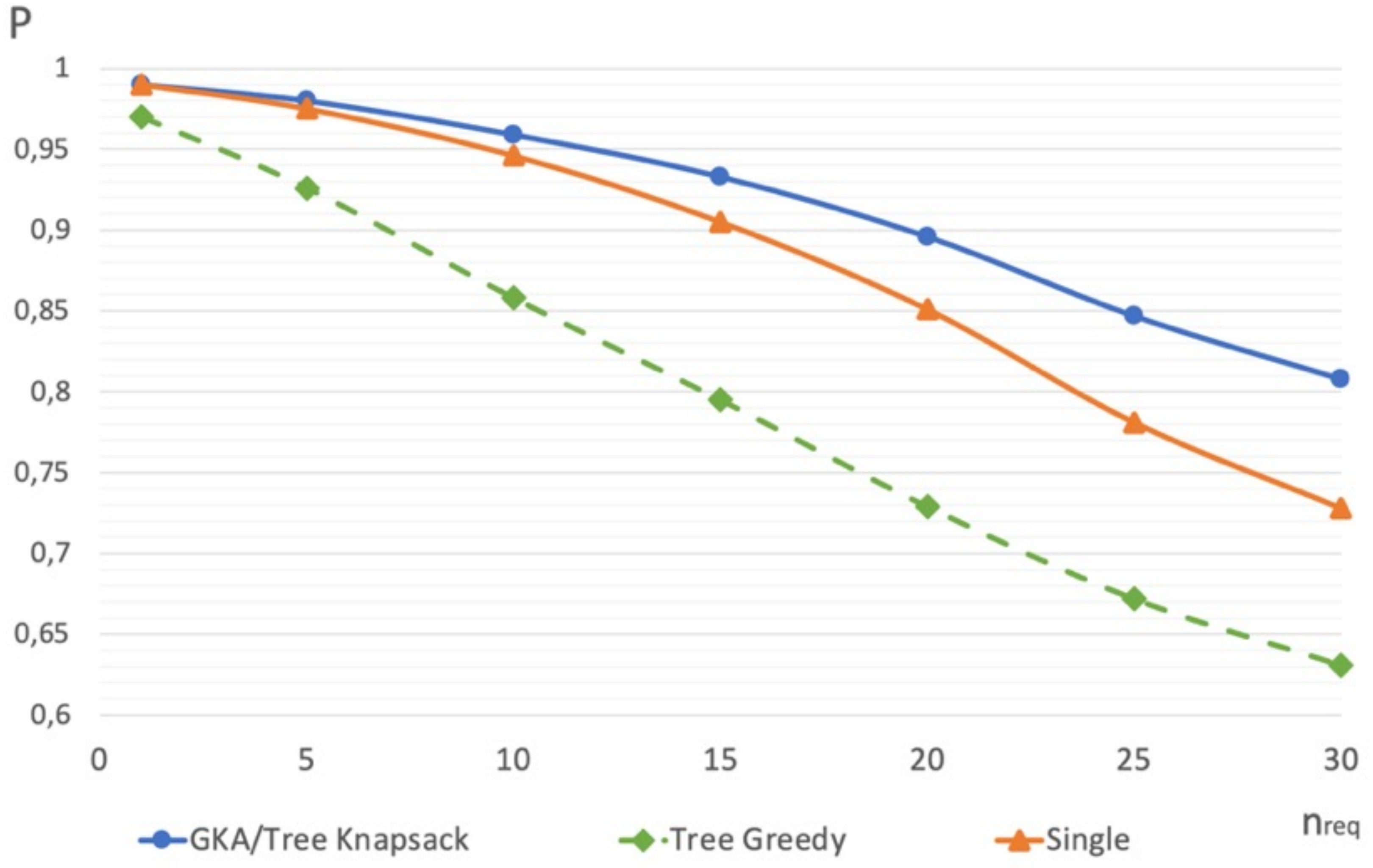

Brute Force, GKA and Branch-n-Bounds generate exact solution

Brute Force execution time shows its combinatorial nature

Brute Force took 10,000 times longer to execute compared to the branch-and-bounds and GKA in a simplified environment with $N = 22$ resources

# Availability Probability

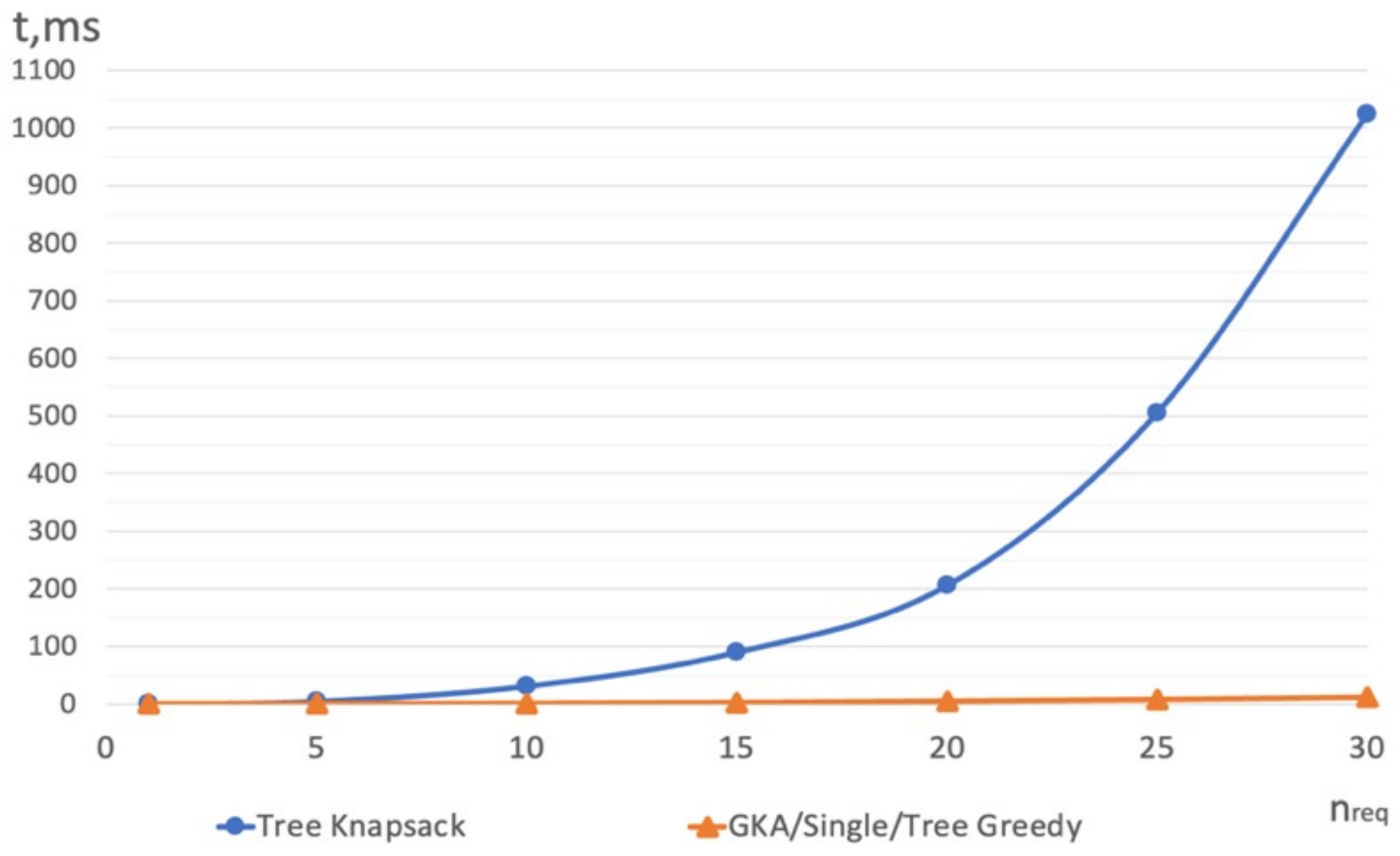
selecting $n \in [1; 30]$ from $N = 200$ resources




GKA and Branch-n-Bounds generate exact solution

Other algorithms provide 5-18% lower values

# Execution Time
## selecting $n \in [1; 30]$ from $N = 200$ resources

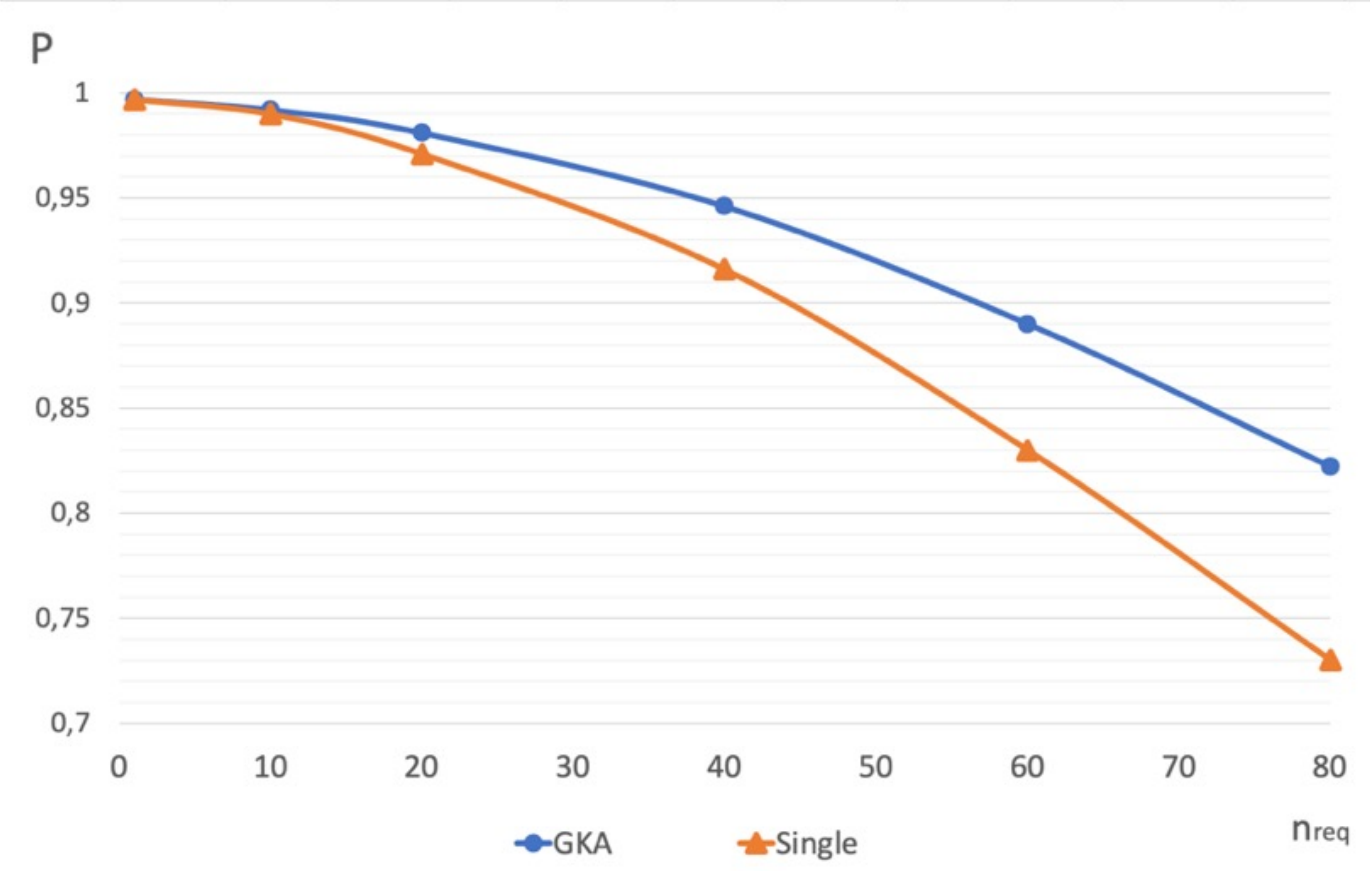



Branch-n-Bounds execution time demonstrates dramatic growth of the decision tree

GKA is at least 100 times faster to solve the same problem

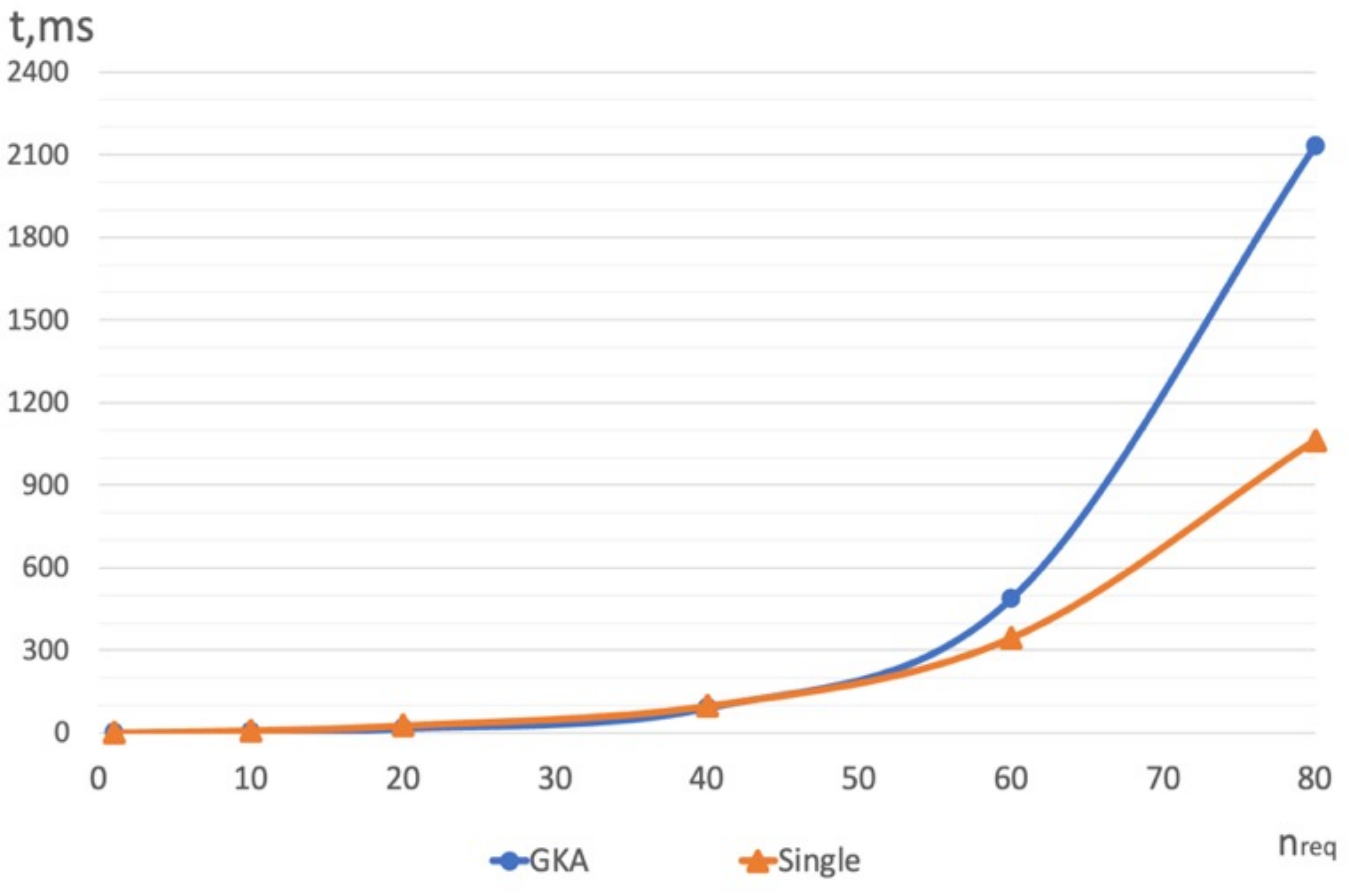
# Availability Probability

selecting $n \in [1;80]$ from $N = 1000$ resources




GKA generates exact solution increasingly better compared to Knapsack

# Execution Time
## selecting $n \in [1; 80]$ from $N = 1000$ resources




GKA and Knapsack execution times and complexity are *comparable*

Algorithms exceed 1 second execution time for the following problem sizes:

Knapsack: $n = 80$ from $N = 1000$

GKA: $n = 65$ from $N = 1000$

BB: $n = 30$ from $N = 200$

BruteForce: $n = 10$ from $N = 22$

# Conclusion

- We address the problem of dependable resources co-allocation for parallel jobs in distributed computing with group dependencies over the resources
- We compared several algorithms and approaches, including brute force, classical knapsack, branch-and-bounds, greedy approximation and a novel dynamic programming procedure
- Proposed solution allows to generate accurate solution for problems with thousands of nodes , (at least 10x times larger compared to branch-and-bounds approach)
- In our further work, we will research possible hybrid approximation schemes and metaheuristics applicable for even larger problems of resources allocation with group dependencies
- This work was supported by the Russian Science Foundation project No. 22-21-00372, https://rscf.ru/en/project/22-21-00372/

Thank You!