Presentation: Using Game Theory to Manage Self-Aware Aerial Systems

Venkata Mandadup and Christopher Stewart, The Ohio State University

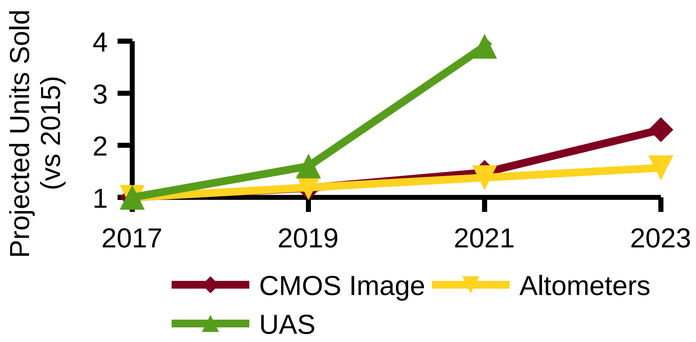
I. Motivation

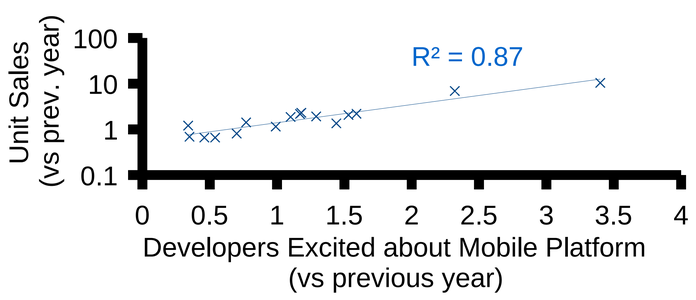
Unmanned aerial systems are increasingly popular

Annual growth rate exceeds that of constituent components.
(a) altometers, image sensors

Manufacturers now provide software development kits (SDK).
(a) DJI support iOS, Android and Linux (Windows coming soon)

Programmable drones are a game changer, enabling a wider range of applications but...





Developers need platform support to build applications.

As shown, unit sales of smart phone operating systems correlates with developer interest.
(a) data take from iOS, Android, Windows and Palm
(b) 2008 – 2012

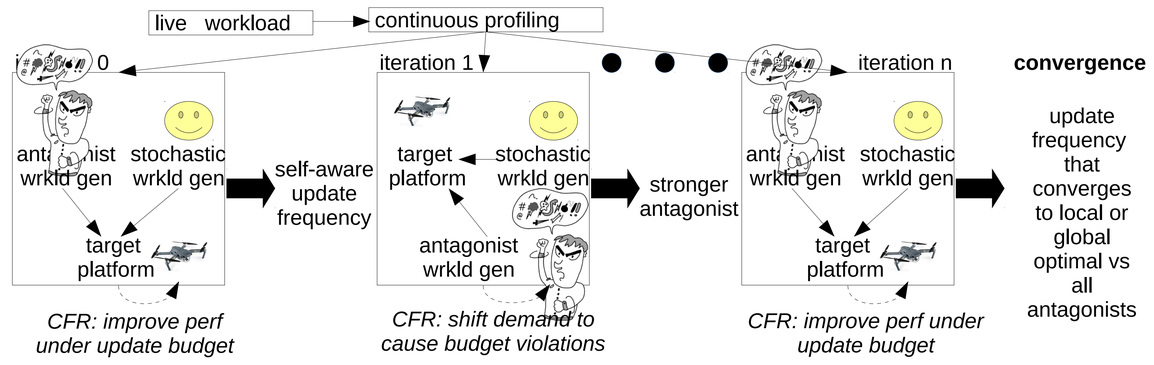
With rapid workload change, UAS platforms must frequently update resource management policies. Research on self-adaptive systems has addressed the technical challenges of:
(1) setting goals for resource management, especially in complex systems,
(2) monitoring policies and assessing efficiency,
(3) choosing between competing policies and models and
(4) efficiently changing policies on the fly.

These approaches often exclude the cost of updating platforms. With rapid workload change, the costs can become prohibitive.

This position paper addresses a problem created by the growth of programmable UAS: *How often should runtime platforms update policies and/or software given limited resources?*

As a running example, we ask the reader to consider software updates, because they require costly programming effort. We argue that game theory approaches can help.

II. Counter-Factual Regret



Our approach:
1. Profile platforms over time: For resource management policies, profiles will assess runtime metrics, e.g., energy efficiency or response time.
2. Develop antagonists: These agents have constrained abilities to affect demand but aim to degrade platform efficiency.
3. Use counter-factual regret: Counter-factual regret is an iterative procedure to minimize the effect of antagonistic forces.

Scenario: Consider two agents that repeatedly compete for resources. Agents act in turns and their actions are influenced by data about prior actions and their effects. Over a long period, each agent would like to make choices that maximize their utility, where utility is a function of resources acquired.

An agent's regret for an action taken is the difference between utility received and utility that could have been received with the best alternative action.

Counterfactual Regret Minimization algorithm[1], [19] :
Decomposes overall regret into multiple additive regret terms. Each player starts with a random strategy for making choices and over multiple iterations updates the strategy based marginal regret of each action. The figure above depicts this approach in the context of update policies.

III. Proposed Design

Problem Statement: Programmers demand high-level programming abstractions. Runtime platforms provide a subset of those abstractions, allowing programmers to invoke them with few lines of code. Programmers must implement the remaining abstractions themselves, requiring many lines of code. Platform developers strive to provide abstractions that minimize lines of code for all programmers, but programmer demands change over time. We model two types of programmers:

1. Typical programmers: Shift their demands according to a known stochastic distribution over time. These shifts reflect predictable changes in the way technology will evolve, e.g., trends in processor speed and battery capacity.

2. Antagonist programmers: Shift their demands to make the platform inefficient. At most a fixed percentage of aggregate demand comes from this group, further game theoretic rules may govern this groups actions, constraining demand shifts from this group.

IV. Early Results

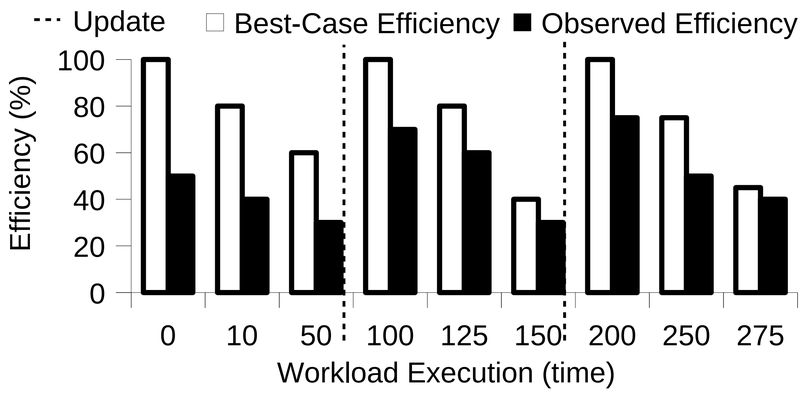
The figure below describes the types of results we target:
(a) Longitudinal studies are key

(b) Best-case efficiency is defined using optimal API (defined by typical and antagonist workloads profiled *before* last update period)

(c) Note, best-case efficiency degrades as workloads change

(d) Observed efficiency is lower than best-case, it includes regret from picking a sub-optimal policy (in hopes of reducing degradation between updates)

(e) Counter-factual regret simulates workload shifts and efficiency results repeatedly changing update frequency until convergence is reached



V. Future work

- Using counter-factual regret to manage realistic workloads. UAS need more, open platforms for such research [11].
- Applying counter-factual regrer to long running workloads that suffer black swan events.
- Rigorous proofs of convergence are needed for applications to UAS, IoT and self-driving cars where mistakes have grave consequences.

VI. References

[1] M. Bowling, N. Burch, M. Johanson, and O. Tammelin. Heads-up limit holdem poker is solved. Science, 347, 2015.
[2] E. Cavalcante, T. Batista, N. Bencomo, and P. Sawyer. Revisiting goal-oriented models for self-aware systems-of-systems. In IEEE International Conference on Autonomic Computing, 2015.
[3] H. Hoffmann and M. Maggio. Pcp: A generalized approach to optimizing performance under power constraints through resource management. In IEEE International Conference on Autonomic Computing, 2014.
[4] S. A. Javadi and A. Gandhi. Dial: Reducing tail latencies for cloud applications via dynamic interference-aware load balancing. In 2017 IEEE International Conference on Autonomic Computing (ICAC), pages 135–144.
[5] J. Kelley, C. Stewart, N. Morris, D. Tiwari, Y. He, and S. Elnikety. Measuring and managing answer quality for online data-intensive services. In IEEE International Conference on Autonomic Computing, 2015.
[6] J. O. Kephart and J. Lenchner. A symbiotic cognitive computing perspective on autonomic computing. In IEEE International Conference on Autonomic Computing, 2015.
[7] S. Kounev, N. Huber, F. Brosig, and X. Zhu. A model-based approach to designing self-aware it systems and infrastructures. Computer, 49(7), 2016.
[8] G. A. Moreno, J. C amara, D. Garlan, and B. Schmerl. Efficient decision-making under uncertainty for proactive self-adaptation. In IEEE International Conference on Autonomic Computing, 2016.
[9] N. Morris, S. M. Renganathan, C. Stewart, R. Birke, and L. Chen. Sprint ability: How well does your software exploit bursts in processing capacity. In IEEE International Conference on Autonomic Computing, 2016.
[10] R. B. Myerson. Refinements of the nash equilibrium concept. International journal of game theory, 7(2), 1978.
[11] J. L. Sanchez-Lopez, R. A. S. Fern andez, H. Bavle, C. Sampedro,M. Molina, J. Pestana, and P. Campoy. Aerostack: An architecture and open-source software framework for aerial robotics. In International Conference on Unmanned Aircraft Systems, 2016.
[12] Stackoverflow.com. Mobile developer surveys, 2013–2018.
[13] C. Stewart, M. Leventi, and K. Shen. Empirical examination of a collaborative web application. In IISWC, 2008.
[14] C. Stewart and K. Shen. Performance modeling and system management for multi-component online services. In USENIX Symposium on Networked Systems Design and Implementation, May 2005.
[15] S. Tomforde, S. Rudolph, K. L. Bellman, and R. P. W urtz. An organic computing perspective on self-improving system interweaving at runtime. In IEEE Conference on Autonomic Computing, 2016.
[16] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, and T. Kelly. Autoparam: Automated control of application-level performance in virtualized server environments. In FeBid, 2007.
[17] D. Weyns and M. U. Iftikhar. Model-based simulation at runtime for self-adaptive systems. In IEEE Conference on Autonomic Computing, 2016.
[18] Z. Xu, N. Deng, C. Stewart, and X. Wang. Cadre: Carbon-aware data replication for geo-diverse services. In IEEE Conference on Autonomic Computing, 2015.
[19] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. In Advances in neural information processing systems, 2008.
[20] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, 2007
[21] C. Stewart, A. Chakrabarti, and R. Griffith, "Zoolander: Efficiently meeting very strict, low-latency SLOs," in International Conference on Autonomic Computing, 2013.
[22] J. Kelley and C. Stewart and N. Morris and D. Tiwari and Y. He and S. Elnikety, "Obtaining and managing answer quality for online data intensive services," in ACM Transactions on Modeling and Performance Evaluation of Computing Systems, 2017.
[23] N. Deng, C. Stewart, D. Gmach, and M. Arlitt. Policy and mechanism for carbon-aware cloud applications. In Proceedings of Network Operations and Management Symposium (NOMS'12). IEEE, 2012.
[24] M.-P. Hosseini, A. Hajisami, and D. Pompili, ''Real-time epileptic seizure detection from eeg signals via random subspace ensemble learning,'' in Proc. IEEE ICAC, Jul. 2016, pp. 209–218.
[25] C. Krupitzer, F. M. Roth, C. Becker, M. Weckesser, M. Lochau, and A. Schurr. Fesas ide: An integrated development environment for autonomic computing. In 2016 IEEE International Conference on Autonomic Computing (ICAC), pages 15–24. IEEE, 2016.
[26] B. H. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, et al. Software engineering for self-adaptive systems: A research roadmap. In Software engineering for self-adaptive systems, pages 1–26. Springer, 2009.